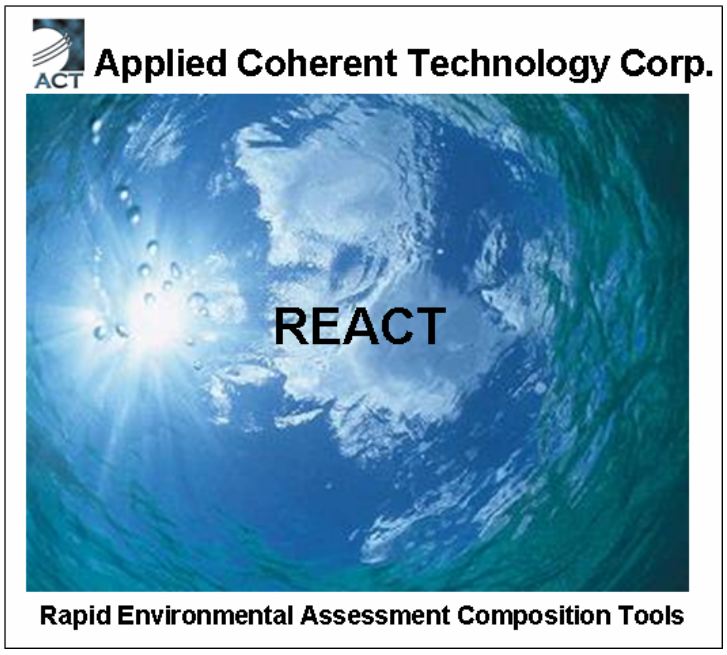
ACT-REACT User's Manual

Part II – MSHELL Development Environment



MSHELL, an image and signal processing language.
© Copyright 1988-2007, Applied Coherent Technology Corp.

ProVIEW, a Professional Virtual Image Processing Environment for Windows.
© Copyright 1994-2007, Applied Coherent Technology Corp.

ACT-REACT, (ACT's Rapid Environmenal Assessment Composition Tools).
© Copyright 2004-2007, Applied Coherent Technology Corp.

GDAL (Geographic Data Abstraction Library) is under copyright by Frank Warmerdam. See appendix

Table of Contents

OVERVIEW **8**

WHAT IS THE REACT? 8
WHO SHOULD USE REACT/MSHELL? 10

REACT'S DEVELOPMENT ENVIRONMENT **11**

REACT ANALYSIS AND DEVELOPMENT ENVIRONMENT 11
Monitoring Panel *11*
Image Window *12*
Script File Window *13*

MSHELL INTERPRETER LANGUAGE **15**

LANGUAGE SYNTAX 15
Introduction *15*
Variable Names and Types *15*
Statements *24*
Calling Syntax for Numeric Functions *24*
Comments Delimiters *25*
Operator Type and Precedence *25*
Region of Interest Manipulation *26*
Program Flow Control and Relational Operators *27*
Look-Up-Table Manipulation *31*
MSHELL SCRIPT FILES (.MSF,.MSH,.VSH) 32
Function Files (.msf) *32*
Include Files (.msh) *35*
Virtual Include Files (.vsh) *35*
IMPORTING AND EXPORTING DATA 36
Importing Data *36*
Exporting Data *37*
INTERNAL FUNCTIONS 38
MATH ERROR HANDLING AND INDEXING 38
System level access *41*
Dynamic Data Exchange *41*
Dynamic Link Libraries (DLL) *43*

APPENDIX A: MSHELL SCRIPTS - EXTERNAL FUNCTIONS TREE **47**

INTRODUCTION 48

APPENDIX B: INTERNAL FUNCTIONS **49**

REACT/MSHELL INTERNAL FUNCTIONS (BY CATEGORY): 50
INTERNAL FUNCTIONS - ALPHABETICAL LIST 55
- Symbols - *55*
- A - *65*
- B - *71*
- C - *75*
- E- *106*
- F - *110*
- G - *126*
- H- *137*
- I - *140*
- L - *153*
- M - *160*
- N - *179*
- O - *185*
- P - *186*
- Q - *198*
- R - *200*
- S - *235*
- T - *279*
- U - *283*
- V - *283*
- W - *292*
- X - *299*
- Z - *304*

APPENDIX C: REMOTE ACCESS 305

INTRODUCTION 305

APPENDIX D: CARTOGRAPHIC LIBRARY 306

PROJ COPYRIGHT NOTICE 306
NAME 307
SYNOPSIS 307
DESCRIPTION 307
EXAMPLE USAGE 309
SEE ALSO 310
HOME PAGE FOR THIS CODE 310

APPENDIX E: GEOGRAPHIC DATA ABSTRACTION LIBRARY 311

GDAL COPYRIGHT NOTICE 311
REACT/MSHELL-GDAL INTERFACE 313
REACT/MSHELL GDAL LIBRARY SUPPORT 314

APPENDIX F: RECENT CHANGES 322

List of Capabilities recently added to REACT/MSHELL 322

Overview

What Is the REACT?

REACT stands for Rapid Environmental Assessment Composition Tools. It provides a number of high level tools, e.g.

- Development Environment,
- Network Centric Data Access,
- Decision Support Tools, and
- Institutional Answers.

This document concentrates on the development and analysis tools of REACT.

REACT is a GUI application with many functionality modules. A core compoment of REACT's development environment is the interface with ACT's MSHELL image/signal processing interpreter language. The REACT/MSHELL GUI interface inherits many of the concepts from ProVIEW[1]*.*

The combination of REACT/MSHELL exposes an interactive command line and menu driven Image and Signal Processing Environment which runs under all major OS environments. The combination of REACT and MSHELL (REACT/MSHELL) provides powerful scientific image and signal processing and visualization capability by affording you:

- Algebraic and matrix operations using mathematically intuitive syntax.
- Support of relational operators and flow control through the built-in ACT's MSHELL image/signal processing interpreter language.
- Double precision image processing computations for high accuracy which support both real and complex number operations.
- Over 300 operators: FFT, convolution, edge detection....

[1] *ProVIEW is no longer supported by ACT and it has been replaced by REACT.*

- Geometric operations: re-size and rotate images using unequal horizontal and vertical scaling.
- The ability to call your own functions as stand alone executables, or Dynamic Link Library (DLL) for MS-Windows implementations.
- The ability to access SQL database engines via ODBC.
- Flexible multiple document interface (MDI) for the display of images, plots, and scripts.
- Contrast processing, linear stretching, intensity range remapping.
- Pseudo Color Lookup Tables for each Image with as many colors as the hardware permits.
- Interactive graphics, i.e. 2D, 3D, and Contour Plots.
- Ability to support multiple image format support: ASCII, 8 bits/pix, floating point, in addition to key standard formats, such as TIFF, BMP, FITS, PGM, PPM, PDS, …
- Interpreter access to many industry standard geo-data-formats via the open-source Geographic Data Abstraction Library (GDAL).
- Ability to project geo raster data to any of the standard USGS cartographic map projections.
- REACT/MSHELL and MSHELL are the software development tool-kit for ACT's: WWW Information Processing Environment (WIPE), and Planetary Information Processing Environment (PIPE).

Expert or novice, independent of your experience, REACT/MSHELL allows you to manipulate images and signals in a simple manner, releasing you from the constant tracking of image attributes such as image dimensionality.

REACT/MSHELL permits you to process a large volume of images in a fully automatic fashion, e.g. large scale reduction, calibration and analysis of satellite based digital images.

If you work with NASA's Planetary Data System (PDS) data, REACT/MSHELL enhances your productivity by providing the following additional capabilities:

- Adoption of the PDS reader libraries
- Virtual Image **variable** which can bigger than 4Gbytes!

Who Should Use REACT/MSHELL?

REACT/MSHELL has been developed by and for professionals working in the areas of image and signal processing who desire to concentrate on algorithmic development and data visualization rather than on low level programming.

REACT/MSHELL can significantly reduce the time required for the development and implementation of image and signal processing algorithms without sacrificing computational performance. A single statement in MSHELL can be equivalent to a large number of statements in other languages, such as C or FORTRAN. REACT's GUI components allows to rapidly visualize the data expeciting the process of algorithm development.

Based on the understanding that both image processing and multi-dimensional signal processing share the same mathematical foundations, MSHELL was developed from the onset as an image and signal processing language. With MSHELL at it's core, **REACT provides an Image and Signal Processing Environment** that is powerful, compact, and simple to use. This makes REACT an excellent tool for work in many diverse application areas, such as:

- Calibration and Reduction of Satellite Imagery, (coherent as well as non-coherent image data),
- Visualization of Multi-Spectral Image Data,
- Modeling of Electro-Optical Imaging Systems,

To effectivly use REACT/MSHELL a working knowledge of Linear Algebra, Image Processing, and Computer Programming is recommended.

REACT's Development Environment

REACT Analysis and Development Environment

To use REACT development environment some familiarity with the Graphical User Interface (GUI) is assumed. This is discussed in more detail in a companion manual that address the GUI and REACT's targeting capabilities.

With REACT/MSHELL you can have multiple images open at the same time, i.e. multiple document interfac. However, you will note that at any given time only one of these windows will be the **Active Window**, i.e. the window with the highlighted top bar. In Figure 1 the Command Shell Window is the active window.

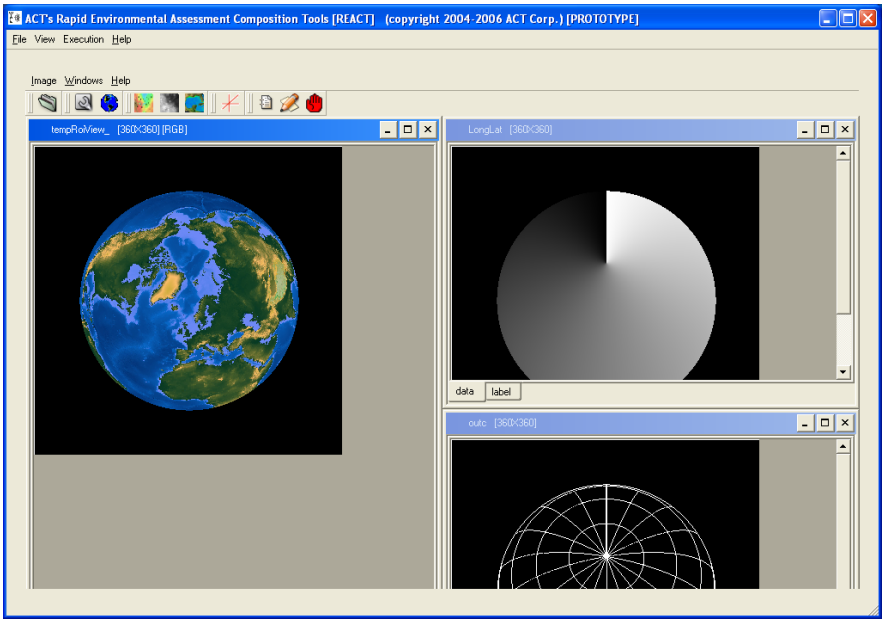


Figure 1 - Sample REACT/MSHELL screen.

In addition, Figure 1 provides examples of some of the different types of REACT Windows: Command Line, Image, Script, and Plot. A description of each of the different . windows follows.

Monitoring Panel

The Monitoring Panel provides a simple way to send single commands to the interpreter, and capture all text output it returns, see right panel on the image below.

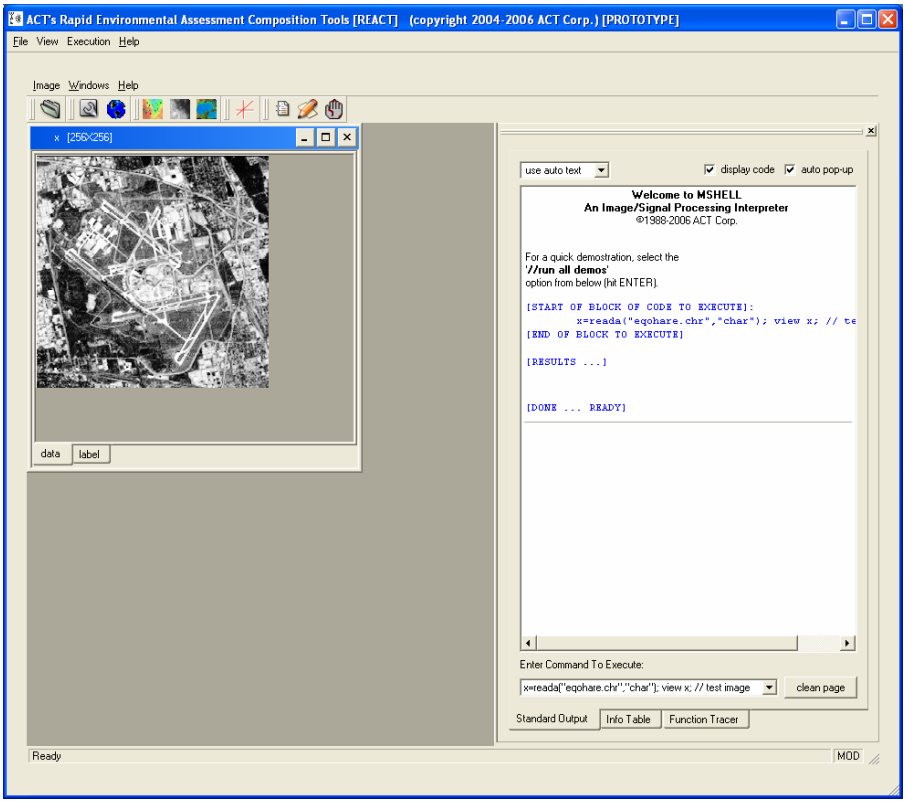


Figure 2 Monitoring Panels is on the right and Multiple Document Interface (MDI) for images on the left.

The user can toggle the view of the Monitoring panel from its corresponding tool bar icon, i.e. . Commands typed in the 'Command To Execute' text region, are submitted and executed by the MSHELL interpreter. It is from within this window that script files are normally invoked, see "**MSHELL Script Files (.msf,.msh,.vsh)**" on page 32. Also, any text output is normally sent to this window. Note that in REACT many of the GUI commands are are converted directly to commands in the "**MSHELL Interpreter Language**", see page 15.

In addition to the Monitoring Panel, the user can send commands to MSHELL using the Scripts Tool Panel, . This will will be discussed in a later section.

Image Window

Prior to enabling the display of an image the user can control many attributes that will affect how the image is to be displayed

An image window provides a visual representation of a two dimensional array of numbers. The simplest way to enable the view of an array is using the 'view' command. In REACT/MSHELL there are several different ways to enable an image window for the display of an existing array variable:

- from the Command Shell Window using the **'view'** command, see **Appendix B**,

- or from the OpenImage toolbar icon, . This will open the OpenImage Panel, i.e.,

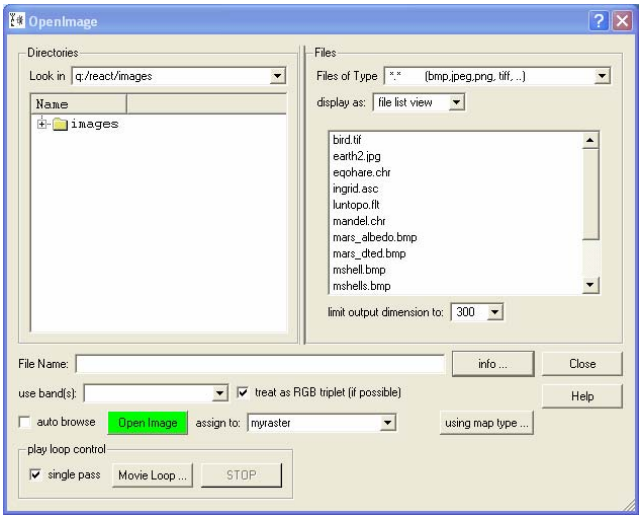


Figure 3 Use this panel to navigate and select an image to open. Many file formats are supported, e.g. JPEG, PNG, TIFF, … , Geo-data-formats, …, NASA's Planetary Data System Data formats.

Once the image is open, you can exercise a good deal of control over how it is displayed. Additionally, you can use the scroll bars that are part of the image window to slide the image horizontally or vertically, or use the cursor to travel over the image. Notice that as you move the cursor over the image, it's row position, column position, and corresponding pixel value are tracked in the status bar at the bottom left of REACT's Main Window. If cartographic or latitude&longitude information is available, it will also be displayed.

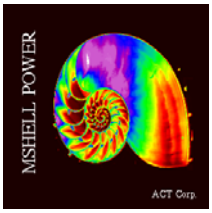
Script File Window

In a Script File Window you can create, edit, or display a text file of MSHELL commands. The easy to use built in editor supports the standard Cut, Paste, Search, and Replace operations found in most windows applications.

Executing a Script File From the Graphical User Interface

To execute the script file located in a Script File Window:

…



MSHELL Interpreter Language

Language Syntax

Introduction

Hint: The system variable M_version can be used to determine if the 32bit or the 64bit version is being executed

At the heart of **REACT** is the **MSHELL** interpreter developed by ACT. MSHELL is a **32** or **64** bit image/signal processing language which allows you to perform complex operations using a simple syntax. In the following sections we will introduce you to this language and discuss it's syntax.

Variable Names and Types

Hint: Use the 'show' command to get a list of all the variables loaded in memory

In general, a variable name can be any alphanumeric string starting with a letter followed by a combination of letters and/or digits. The following are legal alphanumeric variable names,

x,
x10,
OutputImage

Note that variable names should be kept under ***15 characters***. Additionally there are a number of reserved keywords and symbols, used by the interpreter, that cannot be used as variable names. A list of these keywords together with a description of their use is found in the section "**Appendix A** (List of Internal Functions).

There are four basic types of variables:

- Array variables (holding floating point numbers),
- String variables (holding character strings), and
- System variables (used to control the interpreter environment).
- Virtual Variable

Array Variables

The basic variable in MSHELL's interpreter is a two dimensional array structure. More specifically, if '**a**' is the name of an array, then it points to an array structure of the form,

Throughout most of this manual the terms image, array, or matrix can be interchanged without loss of generality

$$a \rightarrow \begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,I-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,I-1} \\ \vdots & \vdots & \cdots & \vdots \\ a_{J-1,0} & a_{J-1,1} & \cdots & a_{J-1,I-1} \end{pmatrix}$$

The upper left element in an image or array is denoted as element (row=0,col=0).

where J is the number of rows in the array, and I is the number of columns. (note: volumetric access is discussed in a following section)

The MSHELL array structure follows the convention that array indices start at zero. Having the basic variable as a two dimensional array provides a unified way to treat scalars, one dimensional signals, and two dimensional signals (images). MSHELL array variables may be either real or complex valued (i.e. hold imaginary numbers). This is particularly advantageous in Fourier transform computations.

The following are valid statements:

```
x = randu(5,10) // generate an array of random numbers

x(0,0)              // extract element value at row=0,column=0

x(1,2:3)  // extract elements 2 and 3, on row 1
```

```
x(:,4:6)          // extract all values between columnes 4&6
                  // and for all rows

x(2,:)            // extract all column values for row 2

y = x(1:3,2:5)    // extract values from row1 to row3 and
                  // column2 to column5, and assigned this to
                  // the array variable 'y'
```

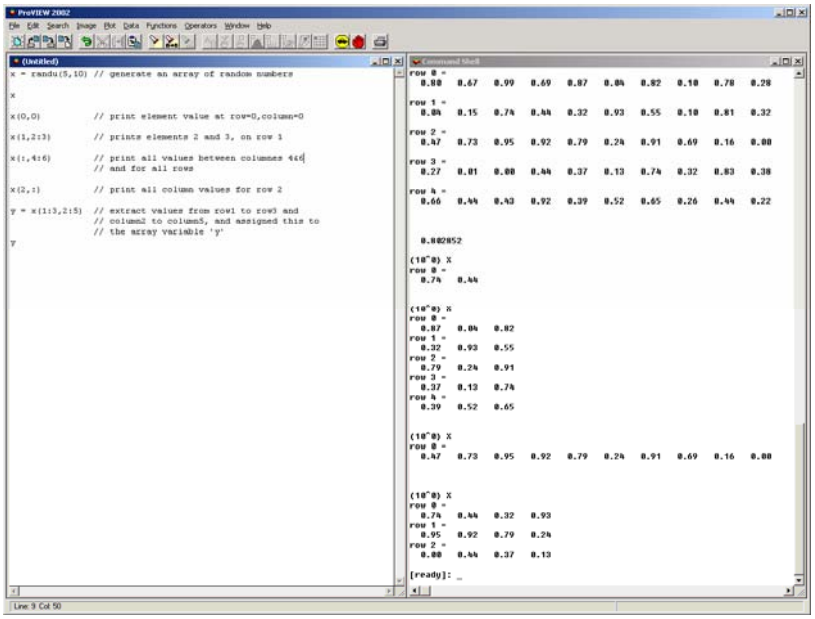

ProVIEW 2002
File Edit Search Image Plot Data Functions Operators Window Help
(Untitled)
x = randu(5,10) // generate an array of random numbers
x
x(0,0) // print element value at row=0,column=0
x(1,2:3) // prints elements 2 and 3, on row 1
x(:,4:6) // print all values between columnes 4&6
// and for all rows
x(2,:) // print all column values for row 2
y = x(1:3,2:5) // extract values from row1 to row3 and
// column2 to column5, and assigned this to
// the array variable 'y'
y
Line: 9 Col: 50
Command Shell
row 0 =
0.88 0.67 0.99 0.69 0.87 0.84 0.82 0.10 0.78 0.28
row 1 =
0.84 0.15 0.74 0.44 0.32 0.93 0.55 0.10 0.81 0.32
row 2 =
0.47 0.73 0.95 0.92 0.79 0.24 0.91 0.69 0.16 0.00
row 3 =
0.27 0.01 0.00 0.44 0.37 0.13 0.74 0.32 0.83 0.38
row 4 =
0.66 0.44 0.43 0.92 0.39 0.52 0.65 0.26 0.44 0.22
0.002852
(10*0) X
row 0 =
0.74 0.44
(10*0) X
row 0 =
0.37 0.84 0.82
row 1 =
0.32 0.93 0.55
row 2 =
0.79 0.24 0.91
row 3 =
0.37 0.13 0.74
row 4 =
0.39 0.52 0.65
(10*0) X
row 0 =
0.47 0.73 0.95 0.92 0.79 0.24 0.91 0.69 0.16 0.00
(10*0) X
row 0 =
0.74 0.44 0.32 0.93
row 1 =
0.95 0.92 0.79 0.24
row 2 =
0.00 0.44 0.37 0.13
[ready]: _

Figure 4 Retrieving array values using row&column ranges.

There are different schemes that facilitate the reading from (or writing to) an array variable. These are discussed in the Region of Interest Manipulation section , on page 26. Note: Included in the region of interest manipulations is a special case that extends the 2D array into a 3D representation for easy visualization and manipulation, which is quite usefull for working with hyperspectral data. For formatted printing of array values see the 'float2str' function.

Intrinsic Attributes Associated with Array Variables

There are many intrinsic attributes associated with array variables. For example, these attributes can control the way a variable is displayed, and interpolated. Most of these attributes can be inspected and changed by the user. The following presents all the intrinsic attributes associated with array variables and their associated syntax. Controlling these attributes can give you finer control.

You may not need to use them since default values are normally applied. But for completeness, let us denote '**x**' as an existing array variable. Then:

x.m_interpflag (This function is not yet implemented.) Selects the type of interpolation to be performed while accessing an element in an array variable by setting:

x.m_interpflag=0 for zero order interpolation.

x.m_interpflag=1 for liner interpolation along the values of a row.

x.m_interpflag=2 for bi-linear interpolation when trying to access the values in an array.

For example, in the following two lines of code the interpolation flag is set to 2 resulting in a request to use bi-linear interpolation.

```
x.m_interpflag=2;

y = x(10.3, 12.7)
```

You can read the value of this attribute.

x.m_viewflag Controls if an image is visible for display or not by setting:

x.m_viewFlag=1 forces the image to be displayed.

x.m_viewFlag=0 disables the display of the image.

Note that the default value for this atribute is zero, not to display the variable.

x.m_viewheight Controls the height of the display window.

x.m_viewwidth Controls the width of the display window.

x.m_viewhscrollpos Controls the horizontal scroll position of the display window.

x.m_viewvscrollpos Controls the vertical scroll position of the display window.

x.m_viewx0 Controls the horizontal position of the upper left corner on the display.

x.m_viewy0 Controls the vertical position of the upper left corner on the display.

x.m_viewlut Contains the active look-up-table to be used for '**x**'.

x.m_viewmaxval Controls the maximum value to be displayed on the screen.

x.m_viewminval Controls the minimum value to be displayed on the screen.

x.m_viewtext Allows one to view the text which is part of an image.

x.text Allows you to access (either read or set) the text attribute of the image.

Image variables may contain associated text, which can be accessed by adding '.text' to the variable name. For exaple,

```
x=4;
x.text="my number"
print x.text ::"\n"
```

x.vroi Extracts the coordinates of the variable region of interest.

Every image variable has associated with it a rectangular region of interest. When a variable is created this region of interest is set to be the whole image. The coordinates of the defined region of interest can be easily accessed at the command line by appending '.vroi' to the image name. For example, the following line of code extracts the coordinates that define the variable region of interest (vroi) of an already defined variable, say X, and assigns it to a user defined variable called 'regionc',

```
[ready]: regionc = X.vroi
```

The notation X.vroi can be viewed as if vroi is an attribute of X.

x.aoi allows access to the actual pixel values defined by the region of interest associated with an array.

Given an array variable, say X, the actual pixel values can be easily accessed in the command line by appending '.aoi' to the image name. For example,

```
[ready]: subimage = X.aoi
```

extracts the subimage defined by X.vroi and assigns it to a user defined variable called 'subimage'. This could also be done using the following notation

[ready]: subimage = X(X.vroi)

MSHELL provides different ways to operate over regions of interest. For example, to add 10 to all pixels falling within the defined region of interest, and updating the display type

[ready]: X(X.vroi) = X.aoi + 10

[ready]: view X

This could also be done as follows,

[ready]: regionc = X.vroi

[ready]: X(regionc) = X(regionc)+10

[ready]: view X

.

The following set of image attributes are related to image mensuration. They all start with '.m_' just as the above intrinsic attribute commands did.

x.m_x0 user defined horizontal position of upper left pixel in image

x.m_dx user defined spacing between two adjacent horizontal pixels as you move from left to right**x.m_y0** user defined vertical position of upper left pixel in image

x.m_dy user defined spacing between two adjacent pixels in the same column as you move from top to bottom

x.m_xunit string describing the units of x.m_x0

x.m_yunit string describing the units of x.m_y0

x.m_flag if this flag is set to 1 the user defined image mensuration will be used when viewing the 'x' image

Volumetric Access of an Array Variable

The following attributes are related to the use of alternative volumetric representation of the data loaded in memory.

x.m_voltype describes the alternative volume type representation to be used for visualization. This variable is read only. Its value can only be modified by the function

	alternateRGB( ) and alternateBSQ( ). voltype=0 ➔ normal 2d array representation voltype=1 ➔ band sequential representation voltype=3 ➔ RGB band sequential
x.m_voldim	similar use to x.m_voltype. It actually contains a string with the Nrows,Ncols, and Nbands to be used in the alternative representation. See the example of alternateBSQ( ) and alternateRGB( ) to undertand the volume type representation better. To read the voldim information use: x.m_nrows3d, x.m_ncols3d, and x.m_nbands3d .
x.m_nrows3d	when the x.m_voltype!=0, this call will return the effective number of rows in the volumetric cube
x.m_ncols3d	when the x.m_voltype!=0, this call will return the effective number of columns in the volumetric cube
x.m_nbands3d	when the x.m_voltype!=0, this call will return the effective number of bands in the volumetric cube

Once the alternateRGB or alternateBSQ representation are enabled, the user can access, a rectangular region of interest, roi, using the following notation,

x(roi| iband)

where,

- iband is a non-negative integer value gving the desired band number, starting at zero.
- roi is a valid rectangular region of interest within a single band.
- This construct can be used for reading. Assignments needs to be implemented.

x(: | iband)

where, provides a short notation to select the whole band, i.e.

- The ':' delimiter is used in this particular example to denote the whole set of rows and columns in the selected image plane.
- iband is a non-negative integer value gving the desired band number, starting at zero.
- This construct can be used for reading and assignments.

See example code in alternateBSQ( ).

String Variables

A string is defined as a sequence of alpha numeric characters enclosed within quotes (similar to the 'C' language).

In general, string variable names start with '**$**'. For example, the string variable '$message' can be assigned a string as follows,

$message = "hello world";

MSHELL allows the use of control characters within a string, such as

\n	linefeed
\t	tabulation
\b	backspace
\\	backslash
\r	carriage return

The above control characters can be used to control the format of strings on the output.

If $x is a string, and r1,r2 are row numbers, and c1,c2 are column numbers, its content can be accessed using the following syntax

```
$x(c1:c2)     // select characters between column c1 and c2 ,
              // independent of any linefeeds or row number.  This
              // notation is particularly fast in execution since it does
              // not requires to make two passes over the string.

$x(r1, c1::c2)        // select characters between columns c1 and c2 in row r1

                $x(:, c1:c2)     // select all characters in all rows between columns
                                 //  c1&c2

                $x(r1,:)         // select all characters in row # r1


                If 'a' is a row vector, say x=(0,3,1), then

$x(a,: )      // extracts all the lines specified by the list in a

$x(a,c1:c2)   // extracts all the lines specified by the list in a
```

// but only the characters between column c1&c2.

Note: Relational Operations are permitted on strings. See Program Flow Control for more info.

System Variables

The majority of the system variables are used for plotting purposes by the 'plot' and 'plot3d' functions. Others are used to control the behaviour of REACT/MSHELL and to hold basic system configuration information. A complete list of these variables can be found in the dictionary of internal functions under M_.

Some of the REACT/MSHELL system variables are strings while others are numbers. All system variables are prefixed by 'M_'. String system variables do not require the '$', they are a special case. For example, to initialize the x-axis label to the string "Time" use

M_xlabel="Time";

Virtual Variable 'V'

MSHELL has a special variable, '**V**', called the virtual variable. With this variable the user can manipulate an image file which can be as large as 2^64 pixels! It supports many formats: 1bytes/pixel, floating point, double precision, unsigned inteter, …

If the user has a huge image in a file or is going to be working with an image that can not be easily held in memory, then he or she can still manipulate pieces of the large image using MSHELL's virtual variable.

Only one image file can be associated at any given time with the virtual variable V. The basic virtual related functions in MSHELL are Vopen, Vclose, and Vnew().

Vopen - Links 'V' to an image file,

Vclose - Stops link between '**V**' and a disk file,

Vnew - Creates a new virtual file.

*See **Appendix B** for detailed usage of the above virtual variable manipulatory functions.*

Once a link is established between a file in disk and the virtual variable '**V**', then the user can access rectangular regions of interest in the disk file for read or write operations (the user must always provide a rectangular region of interest when writing or reading from '**V**').

Statements

A statement is, in general, an expression involving variables, internal functions, and calls to script functions. Multiple statements can be typed in the same physical line if they are separated by a semicolon. For example,

[ready]: x=4; y = x+4; z=cos(y);

is a legal statement.

A statement may expand over more than one physical line if a delimiter '\\' is used to indicate a continuation line, e.g.

```
x =   cos(x+y)    \\
              +   sin(x-y)
```

Tabs, spaces, and linefeeds are ignored by the interpreter. An intuitive mathematical syntax is followed by the interpreter, allowing you to input the expressions in a form similar to the their actual mathematical representation.

In general, expressions which do not involve an assignment will print the result to the screen, e.g.,

```
[ready]: 3+4
7
```

A broad representation of numbers consistent with most computer languages is allowed. The following are valid number representations:

4 0.1 44. 0.44 34.89E4 34.89e10

Calling Syntax for Numeric Functions

The output of most of the numeric functions can be used as direct inputs to other numeric function, e.g.

X = abs(log10(fft2(x)))

Notice that you do not have to worry about the implicit dimensions of x and if it is either real or complex. If a function can not handle a complex input it will return an error message.

Unary Numeric Function Syntax

In general, when using a function with only one input argument, a MSHELL unary function, the following two statements are equivalent,

ufun(varname)

varname.ufun

For example, if x is an array variable the following two lines are equivalent,

y = cos(mean(x)) same as
y = cos(x.mean)

The notation in the second line allows us to look at the mean value as an attribute of x. If the same computation is needed multiple times, and it is over a large array, then it is better to same the result in an intermediate variable. For example,

y = ltclipto(x,x.max, x.max/2)

can be more efficiently written as,

xtheshold= x.max/2; y = ltclipto(x,xthreshold, xthreshold)

Comments Delimiters

Statements enclosed between **/*** and ***/** are ignored by the interpreter. This is used to place comments in a script file or to prevent the execution of the line or lines between the comment delimiters.

For example, a valid group of statements that make use of comment delimiters is,

```
/* Compute the magnitude of the FFT
     on image  x
*/
x = abs(fft2(x))
```

The delimiter '**//**' can be used to create single line comments in a script file, e.g.

```
x = fft2(x)    // note: compute the 2D FFT of x
```

Operator Type and Precedence

The following operator symbols are defined within MSHELL. Most of them are used in algebraic operations. The precedence of operators increases as we move down the list, with operators within the same line having the same precedence.

=	assignment operator
::	row augmentation or string concatenation
#	column augmentation
+ -	array addition and subtraction
*** / *. /.**	multiplication and division (elemental)
^	a^b, "a to the power of b"
-	unary minus, e.g. -x
'	transpose operator, e.g. x'
f	an internal function, e.g. cos()

Hence, in the expression

```
c = a+b*x;
```

the multiplication is performed prior to the addition.

Proper use of parenthesis can result in a significantly faster execution

The user can use parenthesis to force grouping of terms and override the operator precedence. The use of parenthesis for grouping can result in faster code execution. For example, consider the following two expressions,

```
x = (a/b)*c;
x = a*(c/b);
```

if *a* and *b* are scalars, and *c* is a large 2 dimensional or 1dimensional array, the grouping, *(a/b)*c* , executes significantly faster since it involves fewer operations.

In the implementation of array assignments, MSHELL uses the concept of **copy-on-write** (sometimes referred to as "COW"). Copy-on-write is a memory optimization strategy. The fundamental idea is that multiple assignments, say 'x=y; q=y' are pointing to the same location of memory used by 'y'. This pointing scheme is maintained until an operation tries to modify any element in an array, say 'x(0,0)=3', that it is being shared. At this point a private copy of the memory pointed by 'x' is first created then modify to prevent the changes affecting the other arrays. All of this happens transparently to the user. The primary advantage is that memory usage is maintained low until it is absolutely necessary to allocate more memory.

Region of Interest Manipulation

MSHELL supports two types of regions of interest: rectangular regions of interest (**ROI**), and non-connected regions of interest, also called generalized regions of interest (**GROI**). ROIs provide a simple way to refer and access rectangular regions within an image, while GROIs provide a simple way to refer to a list of pixels in an image as an entity.

ROI

A **rectangular region of interest** can be constructed in the following ways:

- Using the window definition function, e.g.

  roi = wdef(row0, col0, nrows, ncols);

- Interactively, using the mouse. This option is selected using the IMAGE|SET-ROI menu option.

If the **ROI** is a valid region of interest, it can be used as an argument of a MSHELL variable. Regions of interest can be specified in any of the following operations,

- Assignments,

  x(wmove(roi,25,40)) = x(roi);

- Within expressions,

  x(roi) = y(roi)+z(roi)-q(roi);

GROI

Generalized regions of interest provide a powerful syntax to perform operations on pixels that do not fall within a rectangular region in an image. A number of functions can return generalized regions of interest, e.g., gtindex, ltindex, and eqindex, finite_index, nan_index, …. Once a generalized region of interest has been defined it can be used as an argument in array expressions.

Example 1.

The following statements will set all the pixels in an image X which fall within the values of 100 and 110 to 0.

zroi = rindex(X,100,110);

X(zroi) = 0;

Example 2.

The following statements will compute the mean value of all those pixels in an image X which have an amplitude less than or equal to the brightest pixel in X.

```
zroi = ltindex(X,X.max);
mean(X(zroi))
```

Generalized region of interest allows for the manipulation of disjoint regions of interest in an image. In this method the pixel coordinates are stored as a complex row vector. The length of the vector corresponds to the number of pixels identified within the generalized region of interest. The real part of the row vector serves as a column index, while the complex part serves as a row index.

Program Flow Control and Relational Operators

The **if**, **if-else**, and **while** statements are used in MSHELL to alter the flow within a script file or to cause iteration. The range of each of these statements is a compound statement consisting of statements enclosed in brackets.

IF Statement

if(expression){

statements

}

if(expression){

statements

} else {

statements

}

The **if** statement causes execution of its compound statements if and only if the relation in the expression results in a non zero value.

The **if-else** has two groups of compound statements. If the relation in the expression returns a non zero value , the first group of statements is executed otherwise the second group of statements is executed.

An early exit from an 'if" block can be performed using the 'ifbreak' statement.

While Statement

while(expression){

statements

}

The **while** statement causes repeated execution of its statements as long as the expression results in a non zero value. The relation is tested before each execution of its range and if the relations is false, control passes to the next statement beyond the range of the while statement.

The traditional '**for**' statement used in the C language, i.e.

for(expression1 ; relation ; expression2){

statements

}

can be constructed using the **while** statement as:

expression1

while(relation){

statements

expression2

}

An early exit from a while loop can be performed using the 'wbreak' statement.

Control Expression with Numeric Values

The statements in an '**if**" or '**while**' block are executed depending on the value returned by the control expression. The control expression can include relational, equality, and logical expressions.

Always use parenthesis when using relational, logical, or equality operators. Do not assume a specific precedence.

Relational, logical, and equality operators all have the same precedence. Hence, parenthesis should be used to obtain the desired groupings. For example, in the following statement there is no ambiguity due to the use of parenthesis,

```
if( (i==4.8) && (p<3) ){
    x=y ;
}
```

Relational Operators

The expression syntax when using a **relational operator** is:

expression1 rel-op expression2

where the relational operators are:

<	less than
>	greater than
<=	less or equal to
>=	greater or equal to

Equality Operators

The expression syntax when using an **equality operator** is:

expression1 equality-op expression2

where the equality operators are:

== test for equality

!= test for inequality

Logical Operators

The expression syntax when using a **logical operator** is:

expression1 logical-op expression2

where the logical operators are:

&& logical and

|| logical or

Control Expression with Strings

The following are the valid comparisons that can be done with strings:

```
$s1==$s2

$s1!=$s2

$s1<$s2

$s1>$s2
```

Remember that these type of comparisons are only meaningfull if used within the control expression in a '**while**' or '**if**' statement. For example, the following code performs a test on the string $a and displays itself as an image depending on the result of the test:

```
$a="hello"

if($a=="hello"){
```

```
        x = text2image($a);

        view x;

    }
```

Look-Up-Table Manipulation

Look-up-tables are primarily used to map intensity values displayed on the screen. Output LUTs have 256 entries numbered 0 through 255 corresponding to the 256 possible different intensity values or colors.

Output LUTs

Before an image in memory is displayed on the screen, it goes through an output LUT. The image's pixel values are used as indices into the output look-up-table, the output of which is displayed on the screen. Image data is transformed through the currently selected output LUT whenever an image is being displayed on the screen. Using the 'select' command, the user can select which is the active output look-up-table. For example, to select **olut3** as the active LUT use the following command,

```
select wolut3
```

The user can read the first 3 output look-up-tables. For example,

```
x= wcolut0;    /* copy olut0 into x   */
```

Inspecting x will show that it corresponds to 3 rows of data, each row with 256 entries corresponding to the red, green, and blue components of the pseudo-color LUT.

Look up table **wcolut4** is a user defined output look-up-table, e.g.

```
/* wcolut3 is set to the histogram equalization
  LUT of image X
*/
wcolut4 = heqlut(X);
```

The following lines modify **wcolut3** to provide intensities of red:

```
/* initialize wcolut4 */
red = (0,255,1);
green = (0,255,1);
blue  = (0,255,1);
wcolut4 = red # 0*green # 0*blue;
```

MSHELL Script Files (.msf,.msh,.vsh)

Script files provide a powerful tool to achieve repetitive processing tasks in a simple manner.

There are two types of script files in MSHELL, i.e. FUNCTION files and Include files.

Function Files (.msf)

A MSHELL function file, or external function, permits the user to define its own functions using the MSHELL language syntax. Local variables defined within a FUNCTION file are automatically erased by the MSHELL interpreter when the script file is finished.

Syntax

A valid MSHELL function must follow the following rules:

- The instructions of a MSHELL function must be stored in a file with the same name as the function, e.g. if the function name was MYFUNC, then the file name must be MYFUNC.MSF. Notice the addition of the '**.msf**' extension.
- Only one function per file.
- Array and string variables can be used as input.
- A function can only return Array variables (at least one must be returned). Strings can be returned in the '.text' attribute of an array variable.
- If a variable is not defined at the present function level the interpreter **will not** look for it at the previous function level. The exception to this are system variables, M_, which are treated as global.
- The first eight characters in the file must be the string FUNCTION follow by the list of output arguments, equated to the function name, and followed by the list of input arguments.

The following is a simple MSHELL function file.

```
FUNCTION [out1,out2]=MYFUNC[in1,in2,in3,$in4]
/* This is a dummy function */

a = 10*in1;   b = 3*in2;
print “The string passed was “::$in4 ::”\n”
out1 = in1-in2
out2 = in1*.in2*.in3 - a + b
```

Notice that the variables 'a' and 'b' in the above example only exist during the duration of subroutine.

Variables needed within the function must be passed to the function and new variables not implicitly declared in the calling statement will be automatically declared as **LOCAL** variables.

An actual call to the above example function could be,

```
x= hammiw(32,32)
y = gtclipto(x,0.5,0.5)
z = x-y
[res1, res2 ] = MYFUNC[ x,y,z]
```

where res1 and res2 will contain the result to the external function call.

It is possible to embed example code in a function to describe its usage better. The following block of code demonstrates this.

```
FUNCTION [out1,out2]=myfunc333[x,$str]
/*
// This is an example function.
// To test look at the behaviour of the interpreter when
passing information
// into a function or making copies of arrays.
// EXAMPLE START

          $mystring = "hello world"
          x=hammiw(256,256)*255   // test image
          x.text="hola"           // initialize .text
attribute

          [y,z]=myfunc333[x,$mystring]    // call the test
function

          print " back from function call:
x(0,0)="::float2str(x(0,0))::"\n"
          print y.text::"\n"  // print text to screen
          view y          // view variable as image
          y(0,0)=255      // modify content of y ...
                          // this trigger a deep-copy of the
values in x, follow by a
                          // modification of the values in y
                          // To see update in screen user
must trigger screen update, e.g. view y
          view y
          view    z
          q=y
          print "q.text (right after the q=y assignment)
==>'"::q.text::"'\n"
          q(1,1)=0    // the element change triggers q to
make its own deep copy of the array values
          print "q.text (right after q(1,1) was set to 0)
==>"::q.text::"\n"
          // note that above … the text attribute is not
copied in a deep copy

// EXAMPLE END
*/

print "within fuction:  x.text = "::x.text::"  was received
from the caller\n"

out1 = x          // make a copy of x.  In actuality this is
a copy-on-write allocation.
                      // MSHELL uses copy-on-write (COW) to
save memory.
                      // The actual copy happens when one of
the arrays is modified.
                      // For example, see the following block
of code ...
```

```
x(0,0)=255       // mofify value of x(0,0) element within
subroutine
                     // This will trigger a decoupling of
the local x variable
                     // from the 'x' in the caller
application, and
                     // from the local 'out' variable.

if( out1(0,0) != x(0,0) ){
          print "within function:  element out(0,0) was not
modified due to implicit deep copy\n"
}

out1.text = $str ;    // save some text in the out.text
attribute
out2      = x+x;
```

Include Files (.msh)

Include files are not function files, hence they do not erase local variables upon termination. Include files are simpler than MSHELL function files and are not constrained by the FUNCTION rules. The user may think of an include file as a sequence of commands that can be invoked with a simple include call. Typically include files are saved to disk with the extension '.**msh**'.

Virtual Include Files (.vsh)

A modification to the standard **'.msh'** is the **'.vsh'** extension; this allows MSHELL to automatically execute the entire contents of the script when opened using the **Image|Open** menu item or **'reada'** syntax of MSHELL. It is particularly used with 'Virtual' variables.

Importing and Exporting Data

Importing Data

Text or array data can be imported in a number of different ways and supportint a number of industry standard formats. Must of the support for loading array or image data is via the the internal 'reada()' function (see page 203). The following table provide a simple description of all basic loading functions.

Basic loading functions	**Brief Description**
readtext()	Use this function to load a text file into a memory. Note: It also has the nice ability to read files over HTTP.
readf()	This function is used to perform a formatted read of a scalar or a string from a file unit.
reada()	Reads an array or image from disk. Multiple file formats (or modes) are supported. Including a generic one that tries to figures the input data format.
V(),	MSHELL has a special variable, '**V**', called the virtual variable. With this variable the user can manipulate an image file which can be as large as 2^64 pixels! It supports many formats: 1bytes/pixel, floating point, double precision, unsigned inteter, …
fits_...	Use this class of functions to load Flexible Image Tranfer System (FITS) formatted data. This format is used extensibely in the Astronomy community. The reada() function can be then used to read particular data objects.
gdal_...	Use this class of functions to load raster data using the Open-Source Geographic Data Abstraction Library (GDAL). The MSHELL/GDAL interface provides extensive access to many geographic raster data formats, e.g. GEO-Tiff, CEOS, HDF, …. See discussion in Appendix.
ncread()	Use this function to load NetCDF type of data.
pds_...	Use this class of functions to load and manipulate Planetary Data System data headers. The reada() function can also be used to read some PDS data objects.

Exporting Data

MSHELL can write image or array data in a number of different output formats, see the writea() and reada() functions.

Data stored in the MSHELL's **char** and **float** format contains a simple 9 byte header. The first 4 bytes contain the number of rows (unsigned 32bits number), the next 4 bytes contain the number of columns (unsigned 32bits number), and the last byte contains a flag indicating if the data is real or complex (1=complex, 0=real).

Internal Functions

The user can locate a specific function by first looking into the Classes of MSHELL's Built-in Functions where commands are divided into functional areas, see Appendix A.

Given a function name or symbol the user can locate specific information about its use under the alphabetical listing of internal functions, see Appendix B.

MSHELL array expressions can be used as arguments to other MSHELL functions. In most of the following functions the input to the functions are arrays and the output result is another array.

Math Error Handling and Indexing

In MSHELL floating point exceptions generate either results in the form of NOT-A-NUMBER (NAN), infinity (+INF), or minus infinity (-INF). The user must inspect an array for such an occurance to determine if invalid number are present in the array.

The example below illustrates manipulations that result in NAN or +INF or -INF. The user can control how math related errors are reported using the system variable 'M_matherrflag' described under M_ in Appendix B.

```
 [ready]: cos(1e38)
  -NAN
[ready]: 1/0
  +INF
[ready]: -1/0
  -INF
[ready]: 0/0
  -NAN
[ready]: x = 1/0 :: 0/0
[ready]: x(0,0)
  +INF
[ready]: x(0,1)
  -NAN
[ready]: log(0)
  -INF
[ready]: log(-1)
0 + 3.14159i
[ready]: 0^0
  1
[ready]: y = 0::1::2:: (0/0)
[ready]: y
(10^0) X
row 0 =
      0.00       1.00       2.00       -NAN
[ready]: z = 0::1::2:: (1/0)
[ready]: z
(10^0) X
row 0 =
            0.00        1.00       2.00       +INF
[ready]: eqindex(y,1/0)
3 + 0i
```

The mshell index operators can be useful at locating and tracking array elements with valid or not valid numeric values, e.g.

nan_index(), [ONLY way to identify NaN elements]

finite_index(),

ltindex(), leindex(),

eqindex(), neqindex()

gtindex(), geindex()

rindex(), rindexc()

The following instructions can illustrate how to use index type of functions to locate elements in an array that are NaN or finite.

```
z=0           // zero value
show z
minf = -1/0       // - infinity
show minf
nan = -1/0+1/0    // generates a Not a Number
show nan
uno = 1
show uno

v = z::minf::nan::uno // construct a vector
v

print "index(v)       // index to elements that are not
zero\n"
index(v)

print "\n finite_index(v) // index to elements that are
finite\n"
finite_index(v)

print "\n eqindex(v,z)    //index to elements equal to 0 \n"
eqindex(v,z)

print "\n eqindex(v,minf) //index to elements equal to -
infinity \n"
eqindex(v,minf)

print "\n eqindex(v,nan)  //index to elements equal to NaN
\n"
eqindex(v,nan)

print "\n nan_index(v)    // index to Not-a-Number
elements\n"
nan_index(v)

print "\n ltindex(v,uno)  // index to elements less than
one\n"
ltindex(v,uno)

print "\n rindex(v,-1/0,1000) // index to all elements
between minus infinity and 1000\n"
rindex(v,-1/0,1000)

print "\n geindex(v,1)    // index to all elements >=1 \n"
geindex(v,1)
```

Extendibility of the Environment

In addition to the internal funtions included and listed above you can further extend the capabilities of MSHELL by:

- Accessing your own 'C' funtions or by
- Accessing other applications that support Dynamic Data Exchange.

System level access

MSHELL permits the users to trigger the execution of stand alone programs via the **system** command. The use of this functions is discussed in page 262.

Dynamic Data Exchange

MSHELL is capable of communicating with other applications that support dynamic data exchange (DDE). The following MSHELL commands are used to establish communications links with other applications via DDE.

DDEInit - this command is used to start a conversation on a particular topic with the server application. Syntax: chan = DDEInit("Application Name","Topic"). "Application Name" is the name of the application you want to communicate with (Ex. MSAccess), and "Topic" is the name of the particular topic. This command returns a channel number associated with the conversation you are requesting or -1 if the operation fails.

DDETerm - this command terminates a conversation. Syntax: DDETerm(chan). chan is the number returned by DDEInit.

DDEExec - this command is used to send a command to the server application once the DDE conversation is established. Syntax: DDEExec(chan,cmd). chan is the channel number returned by DDEInit, and cmd is a string you want the server to execute. DDEExec will return 0 if the server accepted the command, 1 otherwise.

DDEPoke - this command is used to request the server application to accept an unsolicited data item value. Syntax: DDEPoke(chan,item,val). chan is the channel number returned by DDEInit, item is a string that identifies the data item you wish to send a value for, and val is a string containing the value. DDEpoke will return 0 if the server accepted the data item value, 1 otherwise.

DDEReqS - this command is used to request the server application to provide the value of a data item. Syntax: $var = DDEReqS(chan,$item). $var is a MSHELL string variable, chan is the channel number returned by DDEInit, and item is a string that identifies the data item you are requesting the value of.

DDEReqV - this command is used to request the server application to provide the value of a data item. Syntax: var = DDEReqV(chan,$item). 'var' is a MSHELL array variable, chan is the channel number returned by DDEInit, and item is a string that identifies the data item you are requesting the value of.

Example: Using MSHELL scripting language

```
$path = "M:\\NEWINDEX\\BYACCESS\\"
$database = "CL_0001"

/* Open Database */
chan1 = DDEInit("MSAccess","System")
$command = "[OpenDatabase "::$path::$database::".MDB]"
$command
DDEExec(chan1,$command)
DDETerm(chan1)

/* Query Database */
$topic = $database::";SQL"
chan1 = DDEInit("MSAccess",$topic);
$temp = "SELECT NEW_IMAGENAME";
status= DDEPoke(chan1,"SQLText",$temp);
$temp = " FROM header";
status=DDEPoke(chan1,"SQLText",$temp);
$temp = " WHERE ("
status=DDEPoke(chan1,"SQLText",$temp);
$temp = "([SENSOR_NAME]=\"UVVIS\")
status=DDEPoke(chan1,"SQLText",$temp);
$temp = " AND ([FILTER]=\"B\") AND ([MISSION_PHASE]=\"LUNAR
AND MAPPING\")"
status=DDEPoke(chan1,"SQLText",$temp);
$temp = " AND (([CENTER_LATITUDE]>=14.0) AND
([CENTER_LATITUDE]<=35.0))"
status=DDEPoke(chan1,"SQLText",$temp);
$temp = " AND (([CENTER_LONGITUDE]>=25.0) AND
([CENTER_LONGITUDE]<=30.0))"
status=DDEPoke(chan1,"SQLText",$temp);
$temp = ");"
status=DDEPoke(chan1,"SQLText",$temp);
$res = DDEReqS(chan1,"Data");
if ( strlen($res) > 0 ) {
   $query = $query::"\n"::$res
 }
DDETerm(chan1);
```

Dynamic Link Libraries (DLL)

MSHELL permits the users to provide their own functions via dynamic link libraries. Following are the steps required to make use of this capability.

First, write a C program containing the code you want to execute. Include user.h in the header section; user.h is included as part of the installation. The functions that can be called from MSHELL must conform to the following declaration: ARRAY *fname(ARRAY *, ARRAY *, ARRAY *, ARRAY *, ARRAY *), where ARRAY is a data structure defined in user.h. Note that the declaration requires five ARRAY* inputs but the user need not use the five inputs. The functions the user wants to call from MSHELL must be exportable.

Second, generate a 32-bit dynamic link library. The C compiler/linker you use must be able to generate a 32-bit DLL. Consult your compiler reference manual.

Note: The memory allocation routines that MSHELL uses are contained in mem.dll. If you plan to use memory allocation routines in your code (i.e., malloc, calloc, realloc, free, strdup) it is strongly recommended that you use instead the functions contained in mem.dll (i.e., NewMalloc, NewCalloc, NewRealloc, NewFree, NewStrdup). The file prvmem.h is provided with the MSHELL installation. Include it in your source code if you are going to make use of the mem.dll memory allocation routines. If you are using Microsoft Visual C++, include the import library memvc.lib in your project. If you are using Borland C/C++, include the import library membc.lib in your project. Both memvc.lib and membc.lib are included as part of the MSHELL installation.

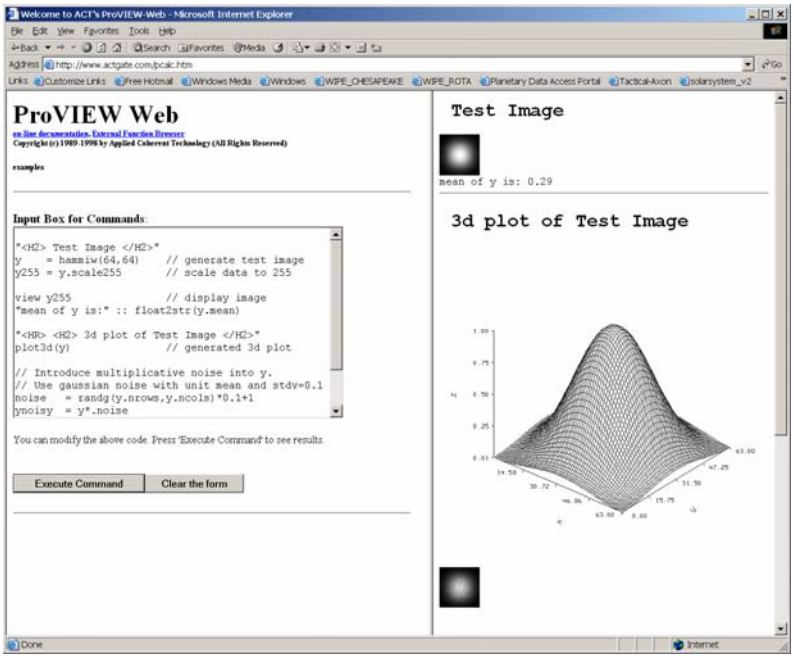
From within MSHELL's Command Shell Window register the DLL using the following command: loadDLL($fullpath\\$dllname), where $fullpath is the path to the DLL including drive letter, and $dllname is the name of the DLL. Note that loadDLL will return 0 if the DLL is successfuly loaded and 1 otherwise.

Call the function in the DLL using the following MSHELL command: callDLL("$dllname","$functionname",arg1,arg2,arg3,arg4,arg5). "$functionname" is the name of the function in the DLL you want to call; arg1, arg2, arg3, arg4 and arg5 are the arguments you want to pass to the function.

```
Example of User Provided Functions as Dynamic Link Libraries

#include <stdio.h>
#include "user.h"

#ifdef __BORLANDC__
ARRAY* _export test(ARRAY *a1, ARRAY *a2, ARRAY *a3, ARRAY
*a4, ARRAY *a5);
#endif

#ifdef _MSC_VER
__declspec( dllexport ) ARRAY* test(ARRAY *a1, ARRAY *a2,
ARRAY *a3, ARRAY *a4, ARRAY *a5);
#endif

/************************************************************
*********/
--- Include the following two functions in your code. MSHELL
will call --- SetInitA and SetFree upon loading the DLL to
pass the addresses of --- the functions inside MSHELL that
InitA and Free will call. It is  --- very important that you
initialize variables of type ARRAY* using  --- InitA. You
must pass to InitA a name you want to assign to
--- the variable, the number of rows, the number of columns,
and the
--- type (0-real, 1-complex).
************************************************************
*********/

void SetInitA(void *fn)
{
          _InitA = (ARRAY* (*)(char *ptr, unsigned long
sizej, unsigned long      sizei, unsigned char type))fn;
}

void SetFree(void *fn)
{
          _Free = (void (*)(ARRAY *a))fn;
}

/************************************************************
*********/

ARRAY* test(ARRAY *a1, ARRAY *a2, ARRAY *a3, ARRAY *a4, ARRAY
*a5)
{
          UN32 i, j;
          ARRAY *c;

          c = InitA("test",a1->sizej,a1->sizei,0);
          if( c )
          {
              for(i=0; i<a1->sizej; i++)
```

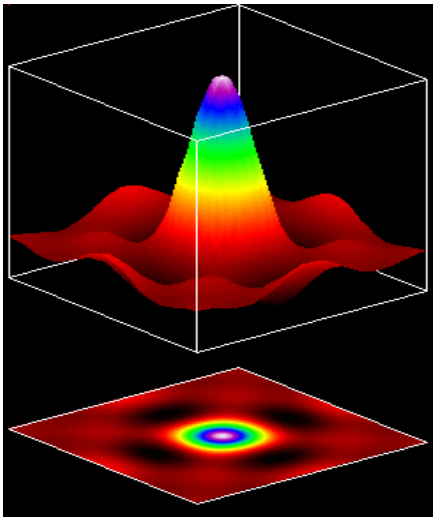
```
            {
                for(j=0; j<a1->sizei; j++)
                {
                    c->re[i][j] = a1->re[i][j] + 10.0;
                }
            }
        }

        return c;
}
```

For illustration a web version of the MSHELL engine can be accessed at: **http://www.actgate.com/pcalc.htm** MSHELL commands can be typed on the left hand side and executed on the server side.


Welcome to ACT's ProVIEW-Web - Microsoft Internet Explorer
ProVIEW Web
Input Box for Commands:
"<H2> Test Image </H2>"
y = hammiw(64,64) // generate test image
y255 = y.scale255 // scale data to 255
view y255 // display image
"mean of y is:" :: float2str(y.mean)
"<HR> <H2> 3d plot of Test Image </H2>"
plot3d(y) // generated 3d plot
// Introduce multiplicative noise into y.
// Use gaussian noise with unit mean and stdv=0.1
noise = randg(y.nrows,y.ncols)*0.1+1
ynoisy = y*.noise
You can modify the above code. Press 'Execute Command' to see results.
Execute Command
Clear the form
Test Image
mean of y is: 0.29
3d plot of Test Image

Note: The file keywords.txt located on the same main directory where MSHELL is residing can be used to specify all commands that remote users are not allowed to execute. This is used as safety measurement.

Appendix A: MSHELL SCRIPTS - External Functions Tree

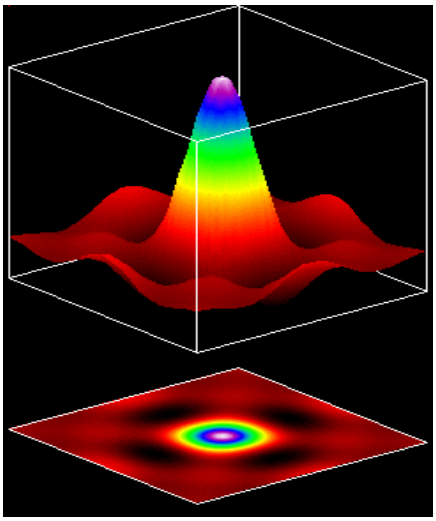


Introduction

With over 140 internal functions and over 100 external functions (scripts), MSHELL is a powerful and flexible environment in which to perform signal and image processing.

The REACT installation tree contains a scripts directory that is broken in subdirectories. The subdirectories contain functional categories. The user is encourages to explore this directory for high level functions implemented in the MSHELL scripting language. All this functions can be tested within REACT from the Analysis panel, i.e.

Appendix B: Internal Functions



(by Category and Alphabetical Listing)

REACT/MSHELL Internal Functions (by Category):

A complete list, by category, of the internal functions is presented on the following pages. See the External functions tree, discussed in Appendix A, for additional functionality.

Bitwise Operators
- and - Bitwise AND operator, 69
- mshift_u16 - Bitwise shifting (16-bits unsigned interger), 177
- mshift_u32 - Bitwise shifting (32-bits unsigned interger), 178

data base operations
- db_functions
  - dbclose - closes access to an external database, 86
  - dbconnect – Connects to database for access, 88
  - dbfopen – Returns a handle to the opened dbf file, 89
  - dbrsgetcolnames – Returns column names in a result set, 92
  - dbrsgetrows – returns result set as a string, 86
  - dbrsgetrowsfloat – Returns the result set as an array, 88
  - dbrsrowcount – returns result set, 86
  - dbsqlexec – Sends a query to the database, 89
  - dbsqltr – Transacts with an external database, 93
- dbf_functions
  - dbfclose – This function closes the dbfile, 90
  - dbfgetfieldcnt – returns the # of fields in the dbf file, 90
  - dbfgetrecordcnt – returns the # of records in the file, 90
  - dbfgetrecords – read the records from the dbf file, 91
- sqlite_functions
  - sqlite_close - Closes access to an SQLite database, 248
  - sqlite_connect – Connects to an SQLite database, 248
  - sqlite_create – Creates an empty SQLite database, 248
  - sqlite_exec – Sends a query to the SQLite database, 249
  - sqlite_getcolnames – Returns column names in a result set, 249
  - sqlite_getrows – Returns the result set as a string data, 249
  - sqlite_getrowsfloat – Returns the result set as an array, 252
  - sqlite_rowcount – Returns the result set count, 253
  - sqlite_transact – Transacts with an SQLite database, 253

Date Operators
- dt2mjd – returns modified Julian Date, 95
- gmtimenow – returns time (y,m,h,m,s), 135
- localtimenow – returns local time, 156
- M_time – see M_ system variables, 163
- mjd2dt – convert modified Julian Date, 176
- tdiff – computes elapse time, 281

Filtering
- Frequency Operations
  - DCT
    - dct8x8 – Discrete Cosine Transform (8x8), 94
    - idct8x8 – Inverse Discrete Cosine Transform, 141
  - FFT
    - fft – One dimensional Fast Fourier Transform, 110
    - fft2 –Two dimensional Fast Fourier Transform, 110
    - ifft – Inverse one dimensional Fast Fourier Transform, 143
    - ifft2 – Inverse two dimensional Fast Fourier Transform, 143
  - Spectral Windows
    - blackw – Blackman-Harris Window, 71
    - hammiw – Hamming Window, 137
- Spatial Operations
  - Correlation Operations
    - convol – Discrete Convolution, 78
    - convoltl – Truncated Discrete Convolution, 79
    - spatf – Spatial Filter, 247
    - xcorr – Cross Correlation, 299
    - xcorrfft – Cross-correlation of two FFT's, 299
    - xcorrt – Truncated Cross Correlation, 299
  - Interpolation
    - Resampling
      - blinterp – Bi-linear Interpolation, 72
      - decimate – Signal Decimation, 94
      - linterp – Linear Interpolation, 154
      - zinterp – Zero Order Interpolation, 304
  - Local Statistics
    - covm – Covariance Matrix Estimation, 81, 82
    - gauss – N-Dimensional Gaussian Density, 126
    - hist – Histogram Generator, 138
    - hist255 – Histogram of 8 bit data, 138
    - qgauss – Area Under Gaussian Density, 198
    - qgaussinv – Inverse of qgauss, 198
    - randg – Gaussian Random # Generator, 200
    - randinit – Random Number Seed Initializer, 200
    - randu – Uniform Random # Generator, 200
  - Morphological
    - img2contour – create contour image, 147
    - skeleton – Binary Conversion Filter, 246

Flow Control and Relational Operators
- // – Single Line Comment, 58
- \\ – continuation line delimeter, 58
- <,>,.. – logical operators, 63
- END – Ends Execution of a Script, 106
- exit – Exit MSHELL/ProVIEW, 108
- IF-ELSE –Conditional Flow, 142
- include – Invoke an MSHELL Script File, 147
- pause – Suspend Execution for N Seconds, 186
- return – Returns From an Include File, 216

stop_process – terminates execution with message, 254
while – While Loop, 294
GUI
addmenuitem – Adds a User Defined Menu Item, 65
Graphical User Interface – Details, 11
iboxlist – Input Box List, 140
inputbox – Prompt User for Input, 150
mbox – Text Box, 173
menusel – creates a Menu List Box, 174
meter – Displays a Metering Toolbar, 174
REACT
react_enable_view – shows a REACT panel, 204
react_get_carto_info - Retrieve Cartographic Viewer Information, 202
react_get_properties- get values for a set of properties, 205
react_get_property – get the value of a REACT property, 205
react_load_mlt – load an MLT file into REACT, 202
react_open_url – Opens an URL in a browser window, 204
react_set_carto_map_info - Set Cartographic Viewer Map Info, 202
react_set_property – set the value of a property, 206
react_update_layers – update layers, 203
view – activate image display, 287
view – activate image display with scaling, 289
viewout – closes image display window, 287
wclose – Closes all Screen Windows, 292
wtile – Tiles all Screen Windows, 298
Image & Volumetric Data Representation
alternateBSQ – describe array as a BSQ volume, 67
alternateRGB – describe array as an RGB volume, 67
Image Text Overlay
text2image – Converts Text to an Image, 280
textoverlay – Annotates an Image with Text, 279
textremove – Removes Text annoation, 280
Image/Array/Matrix/Vector Algebra
' – Matrix Transpose, 60
– Array Subtraction, 56
– Column Augmentation, 62
(a,b,s[,n]) – Generate a 1-D or 2-D ramp, 55
* – Matrix Multiplication, 57
*. – Array Element Multiplication, 57
/ – Array Division by Scalar, 58
/. – Element Array Division, 59
: – Array Interval Delimiter, 62
:: – Concatenate Arrays or Strings, 61
^ – Raise Array Elements to a Power, 59
+ – Array Addition, 56
= – Assignment, 62
abs – Absolute Value, 65
allocate_array - Allocate an array and set its values, 66
ceil – Find the Ceiling of an Array, 75
centroid – centroid of an image, 75
complex – Creates a Complex Array, 77
conj –Array Conjugate, 77
convtoi – Convert Row Vector to Image, 80
convtov – Convert Image to Row Vector, 80
exp – Inverse-Natural Logarithm, 109
floor – Floor of Input Array, 120
fmod – Floating-Point Modulus, 120
free – Free Variable from Memory, 121
geo – Geometric Series, 127
ident – Generate an Identity Array, 142
imag – Imaginary Part, 143
int – Integer part, 151
int – Inverse of an Array, 152
log – Natural Logarithm, 157
log10 – Base 10 Logarithm, 157
m_ – Array attributes, 167
makecmplx – converts a real array to complex, 167
ncols – Number of Columns, 181
nint – Nearest Integer, 184
nrows – Number of Rows, 184
ones – Initialize an Array to all ones, 185
real – Real Part of an Array, 215
show – Display Variables Information, 244
sign – Sign of Array Elements, 245
sqrt – Square Root, 254
sum – Sum All Elements, 260
sumc – Sum Column Vectors in an Array, 260
sumcum – Row Wise Cumulative Sum, 260
sumr – Sum Row Vectors in an Array, 260
svd – Singular Value Decomposition, 261
trace – Sum Diagonal Elements, 282
vartype – Returns the Variable Type, 287
zeropad – Expand an Image with Zeroes, 304
zeros – Initialize Array to all Zeros, 304
Intensity Mapping
Look up Table Operators
heqlut – Histogram Equalization LUT, 137
hyplut – Hyperbolic Histogram, 139
select – Selects an Output Look-Up Table, 236
wcolut[#]– Color Look up Table, 292
wolut[#]– Windows Output Look up Table, 294
xlut[#]–Look-Up-Table Transformation, 301
scale255 – Scale to 8-bit Range, 235
Selected Pixels
bthresh – Binary Threshold, 73
geclipto – Greater or Equal Clip to, 126
gtclipto – Greater than Clip to, 136
leclipto – Lower or Equal Clip to, 154
ltclipto – Lower than Clip to, 158
lthresh – Less than Threshold, 158
IO
Data Formats
FITS
fits_close_file – closes a fits file, 118
fits_get_colnum – Returns the column #, 116
fits_get_coltype – Returns the column type, 117
fits_get_hdu_type – Returns HDU Type, 113
fits_get_num_hdus – # of HDUs, 112
fits_get_num_keywords – # of keywords, 113
fits_get_size – # of rows and columns, 116
fits_movabs_hdu – Moves to a specified HDU, 112
fits_open_file – Open a fits file, 112
fits_read_ keyword_value – Returns Key value, 115
fits_read_col_str – Reads column strings, 117
fits_read_col_values – Reads column values, 117
fits_read_keyn_comment – Returns comment, 115
fits_read_keyn_name – Returns keyname, 114
fits_read_keyn_value – Returns a keyvalue, 114

fits_read_keyword_comment – Returns comment, 115
MSHELL/GDAL– bindings to the GDAL library, 314
NetCDF
ncaddvar – NetCDF Add Variable, 179
ncinq - NetCDF File Inquiry, 179
ncinqvar - NetCDF Variable Inquiry, 180
ncnew - create new NetCDF file, 180
ncread - variable reader, 181
ncvarattstr - Variable Attribute String, 182
ncvarattval - Variable Attribute Value, 182
ncwrite – write variable, 182
ncwriteatt – Write Attribute, 183
PDS
pdsfindobj – Get list of objects, 186
pdskwdstr – PDS keyword value, 186
Shapefile
shp_from_csv_ll – create shapefile from ASCII input file, 237
shp2contour – Shapefile to Contour Image, 238
shp2fillimage – Shapefile Image Fill, 242
shp2image – Shapefile to Image, 240
shp2vector – Retrieve verticies from a shapefile, 242
shpGetInfo – provides info on a shapefile, 243
shpwriteroi – Copy a subset of a shapefile, 243
Virtual
V – Virtual Variable, 283
Vclose – Closes Virtual Variable, 284
Vnew – Makes Disk Space for Virtual Image, 284
Vopen – Makes Virtual Variable Link to File, 285
VPF
vec2image – Vector Format to Image Format, 287
Dynamic Data Exchange
DDEExec - Sends a command to the server app, 95
DDEInit - initialize DDE server application, 96
DDEPoke - POKE DDE server, 96
DDEReqS – Requests value from DDE server, 96
DDEReqV – Requests value from DDE server, 96
DDETerm – close a DDE conversation, 97
Dynamic Link Library
callDLL – Calls a DLL for execution, 75
loadDLL – Loads a DLL for Execution, 155
Formatted
closef – Close a File, 76
openf – Open a file for Formatted I/O, 185
readf – Formatted File Read, 214
writef – Formatted File Write, 297
input – waits user input, 150
load – Loads saved arrays or strings from a file, 155
print – Formatted Print, 196
reada – Read Array or Image, 207
readtext – Loads Text File, 215
readtext_and_status – Loads Text File with check, 215
save – Saves Arrays or Strings to Disk, 235
writea – Write Array to Disk, 295
writecolor – Writes a Color Image, 295, 296
Mirror and Shift Operators
cmirror – Mirrors an Image Column Wise, 76
rmirror – Row Mirror, 218
shiftc – Cyclic Shift of an Image, 237
shiftt – Shift an Array or Image, 243
Plot
colplot – Plots a Row from an Array, 77
contour – Generates a contour plot, 77
dworld – draws world contours, 97
plot – Plot a Vector, 187
plot3d – 3-D Plot or Mesh Plot, 189
polyfill – Fills an Image with Polygons, 190
rowplot – Plots a Row from Array, 223
view4d – project planar image into an xyz grid, 289
Region Ops
aoi – Active Region of Interest, 69
bresen – Compute Line Segment Points, 72
Groi_Logical
regionand – ROI intersection, 216
Index Pixels
eqindex – Equality Index, 106
finite_index – Index of Non-infinite Elements, 112
geindex – Greater than or Equal Index, 127
gtindex – Greater than Index, 136
index – Index of Non-Zero Elements, 150
leindex – Lower or Equal Index, 154
ltindex – Lower Index, 158, 159
nan_index – Index of all NaN Elements, 179
neqindex – non equal index test, 183
rindex – Range Index, 217
rindexc – Range Index Complement, 218
Interactive Selection
inputfocus – Array Variable with Current Focus, 151
setroi – Interactively sets an ROI, 236
varname – Returns the Variable Name, 286
ladd2groi – local add of constant to selected pixels, 153
maskofEQ –mask of elemens where a==b, 167
maskofGE –mask of bit plane, 167
maskofGE –mask of elemens where a>=b, 168
maskofGT –mask of elemens where a>b, 170
maskofLE –mask of elemens where a<=b, 170
maskofLT –mask of elemens where a<b, 170
Overlapping Regions
cmplxoverlap – 2-D range index, 76
pixval – Displays Pixel Status of Mouse, 186
rfill – Fills a subregion, 216
wdef – Define a Region of Interest, 292
wmove – Move a Region of Interest, 294
wsize – Size of a Region of Interest, 298
xline – Extract Pixel Values along Line Segment, 300
xlinec – Extracts Coordinates of Line, 300
xlinev – Extracts Vertices of Line, 301
xpolyc – Extracts Coordinates of a Polygon, 302
xpolyv – Extracts Vertices of a Polygon, 303
Remote Access
url_copy_to_file - Save a remote resource to a file, 283
Remote Access - FTP Protocol Functions
ftp_cd - Changes Current FTP Server Directory, 121
ftp_cwd - Get Current FTP Server Directory, 121
ftp_del - Delete a file in the FTP Server, 121
ftp_get - Download a File from the FTP Server, 122
ftp_list - List FTP Server Files and Subdirectories, 122
ftp_mkdir - Create New Directory in the FTP Server, 123
ftp_put - Upload a File to the FTP Server, 124
ftp_rename - Rename a File in the FTP Server, 124
ftp_rmdir - Remove Directory from the FTP Server, 124
ftp_session_close - Close Current FTP Session, 124
ftp_session_close - Open an FTP Session, 124

Remote Access to MSHELL
- rmshell_execute – Execute a command in the remote MSHELL, 218
- rmshell_execute_async – Execute a command in the remote MSHELL (does not block), 219
- rmshell_getfile – Download a file from the remote MSHELL, 219
- rmshell_getvar – Get an ARRAY variable from the remote MSHELL, 220
- rmshell_getvarS – Get a string variable from the remote MSHELL, 221
- rmshell_putfile – Upload a file from the remote MSHELL, 222
- rmshell_putvar – Put an ARRAY variable to the remote MSHELL, 222
- rmshell_putvarS – Put a string variable to the remote MSHELL, 222
- rmshell_session_close – Close connection to a remote MSHELL interpreter, 222
- rmshell_session_init – Connect to a remote MSHELL interpreter, 223
- rmshell_wait – Wait the number of specified seconds, 219

Remote Access to WIPE (CORBA Based)
- rs_availDataSets – Returns list of available data sets, 224
- rs_availWipeServers – Returns available Server, 224
- rs_connect – connect to a remote WIPE Object, 225
- rs_exec – Executes command in the remote server, 226
- rs_exec_async – Same as rs_exec, 226
- rs_extractData – extract data from the Wipe Server, 224
- rs_getArray – Retrieves the content of an Array, 227
- rs_getMetaData – Extract the metadata request, 225
- rs_getString – Retrieves the content of a string, 227
- rs_init – loads msh32rs.dll, 225
- rs_release – closesconnection with a WIPE Object, 226
- rs_wait – Waits the number of specified seconds, 227

Remote Access to WIPE (SOAP Based)
- rw_getInventoryXML – Retrieve Inventory XML String, 228
- rw_getInventoryXMLAsync – Retrieve Inventory XML String (does not block), 229
- rw_getMosaicXML – Retrieve MOSAIC XML String, 230
- rw_getMosaicXMLAsync – Retrieve MOSAIC XML String (does not block), 231
- rw_waitForInventoryXML – Wait the number of specified seconds, 232
- rw_waitForMosaicXML – Wait the number of specified seconds, 233

Satellite Image Mapping
- Geo_cs2cs – geo coordinate system tranformation, 131
- Geo2Cart – Geographic to Cartographic conversion, 128
- glflyover – generate perspective view of terrrain, 134
- image2surface – project planar image to a surface, 144
- SatVIEW – computes spacecraft geometry info, 235

Shapefile
- polygonorient, 191

Statistics
- mean – Mean of Array Elements, 173
- median – Compute Median Value, 173
- medianr – Row Wise Median, 173
- momentr – Row Wise Moment, 177
- rcoeff – Correlation Coefficient, 201
- Sorting
  - max – Maximum in Array, 171
  - maxmin – Max and Min values in Array, 171
  - maxminx – Max and Min values, with exclusion, 171
  - maxof – Element by Element Maximum, 172
  - maxr – Row Maximum, 172
  - min – Minimum in Array, 175
  - minof – Element by Element Minimum, 175
  - minr – Row Minimum, 176
  - sortr – Row Wise Sorting, 247
- stats – Computes Array Basic Statistics, 254
- statsx – Computes Statistics with exclusion values, 254
- var – Variance, 286

String Operations
- $string – String Access Control, 63
- eqindexS – Equality Index String, 107
- evaltext – Evaluates a String, 108
- expand - Environment variables expansion, 109
- float2str – Convert Array to formatted String, 118
- getline – Finds Line Matching String Pattern, 132
- getpos – Finds String Position Within a Line, 133
- int2str – Convert an Integer to a String, 151
- itoa – Integer to Ascii, 152
- nlines – Returns number of Lines, 184
- Regular Expression Support
  - preg_match – regular expression matching, 191
  - preg_replace – regular expression replace, 195
- smodify – String Replace, 247
- str2float – Converts Numeric String to Float, 255
- str2int – Converts Numeric String to Integer, 256
- strContains – check for substring occurance, 257
- strEndsWith – string termination match, 257
- strhex2ing – Converts Hex string to integer, 258
- strlow2up – Converts lowercase to uppercase, 258
- strSection – extract section of a String, 259
- strStartsWith – string start match, 259
- strStripWhiteSpace – string whitespace strip, 258
- strup2low – Converts uppercase to lowercase, 259
- XML Parsing
  - xmlattrvalues – Extracts the attribute values, 301
  - xmlparse – parse an XML string, 302
  - xmltextvalue – Extracts the text value of a, 302
  - xmltextvalues – Extracts the text values, 302

System
- all – Allows for all Variables to be included, 66
- dirlist – returns list of subdir/files, 94
- diskfreespace – returns disk info, 95
- fileinfo – Returns detailed information of a file, 110
- filesize – Returns the size of a file, 111
- findfiles – Locate Files in Directory Structure, 111
- getenv – Get Environment Variable, 132
- getMyIP – Get the local IP addresses, 132
- help – Invokes the Help Utility, 137
- M_ – System Variables, 160
- putenv – assign to sys. env. variable, 197
- sys_
  - sys_cat, 277
  - sys_cd – changes the current directory, 264
  - sys_copy – copies files, 272
  - sys_del – delete files or empty directories, 274
  - sys_deltree – deletes an entire directory tree, 275
  - sys_dir –directory listing, 267
  - sys_dirlist – controlled directory listing, 267

sys_dirsize –size of a directory tree, 276
sys_filesize – size of a file, 277
sys_mkdir – creates a directory, 266
sys_move – moves or renames files or directories, 269
sys_pwd – current working directory, 263
sys_ren – renames files or directories, 271
sys_sendmail – sends mail to destination, 263
system – Issues Operating System Command, 262
Trigonometric Functions
acos – Inverse Cosine, 65
asin – Inverse Sine, 69
atan – Inverse Tangent, 69
atan2 – Inverse Tangent, 70
cos – Cosine, 81
cosh – Hyperbolic Cosine, 81
sin – Sine, 245
sinc – Sinc Function, 245
sinh – Hyperbolic Sine, 246
tan – Tangent, 279
tanh – Hyperbolic Tangent, 279

Internal Functions - Alphabetical List

Much of REACT power resides in the MSHELL Interpreter. With over 140 internal commands, REACT has a flexible structure that readily permits tailoring to specific applications through User Defined external functions and script files

This appendix is an alphabetical listing with descriptions and examples of all the current **REACT/MSHELL Internal Functions**.

- Symbols -

(a,b,s[,n]) Generate a 1-D or 2-D ramp

Syntax: **(**start_value**,** end_value**,** step_size**)** 1-D
(start_value**,** end_value**,** step_size**,** number_of_rows**)** 2-D

Description: returns a 1-D or 2-D ramp of values

Example: The following applications illustrate the use of the 1-D and 2-D ramp function.

```
[ready]: (0,3,1)          //generate a 1-D row
  0.00   1.00   2.00   3.00
[ready]: (0,1,0.2)           // generate 1-D row with
fractional step size
  0.00   0.20   0.40   0.60   0.80
[ready]: (3,0,1,2)           // generate a 2-D matrix
  3.00   2.00   1.00   0.00
  3.00   2.00   1.00   0.00
[ready]: pi = 3.14159;       // define PI
[ready]: plot( 2 + cos( (0,4*pi, pi/4) ) )   // create a
plot
```

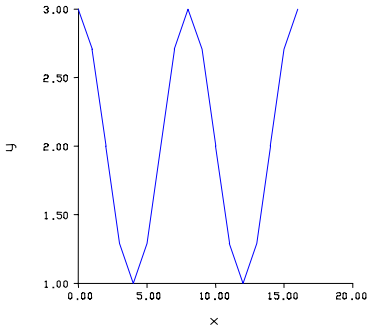


Figure 1

+ *Array Addition*

Syntax: a **+** b

Description: The operator symbol **+** is used to perform the addition of two array expressions. The actual sum is implemented as:

$$c_{j,i} = a_{j,i} + b_{j,i} \quad \text{,for all j, i;}$$

where j and i are, respectively, row and column indices. If 'b' is a scalar and 'a' is not, then 'b' will be added to each of the elements in 'a'.

Example: The following MSHELL statements illustrate simple examples of array addition,

```
[ready]: a = (10,15,1)        // creates row vector 'a'
[ready]: b = a            // copies 'a' into 'b'
[ready]: c = a+b          // add corresponding elements in
'a' and 'b'
[ready]: c                    // prints the 'c' row vector
(10^0) X
row 0 =
 20.00  22.00  24.00  26.00  28.00  30.00
[ready]: c+5              // add 5 to every element in 'c'
(10^0) X
row 0 =
 25.00  27.00  29.00  31.00  33.00  35.00
```

- *Array Subtraction*

Syntax: a **-** b

Description: The operator symbol **-** is used to perform the subtraction of two array expressions. The actual subtraction is implemented as:

$$c_{j,i} = a_{j,i} - b_{j,i} \quad \text{,for all j, i;}$$

where j and i are row and column indices respectively. If 'b' is a scalar and 'a' is not, then 'b' will be subtracted from all the elements in 'a'.

Example: The following MSHELL statement will subtract array 'b' from 'a' and store the result in 'c'.

```
[ready]: c = a-b
```

* *Matrix Multiplication*

Syntax: a * b

Description: The operator symbol * is used to perform the multiplication of two matrices or arrays. The multiplication follows the rules of matrix multiplication used in linear algebra:

$$c_{j,i} = \sum_n a_{j,n} \cdot b_{n,j} \quad \text{,for all j, i;}$$

where j and i are row and column indices respectively. An error message will be generated if the number of columns in 'a' is not equal to the number of rows in 'b'. If 'a' is a scalar and 'b' is not (or vice versa), every element in 'b' will be multiplied by 'a'.

Example: The following MSHELL statements illustrate simple examples of array multiplication,

```
[ready]: a = ones(3,3)        // creates 3 x 3 matrix with
all ones
[ready]: b = ones(1,3)        // creates 1 x 3 vector with
all ones
[ready]: a                    // prints 'a'
row 0 =
  1.00   1.00   1.00
row 1 =
  1.00   1.00   1.00
row 2 =
  1.00   1.00   1.00
[ready]: b                    // prints 'b'
row 0 =
  1.00   1.00   1.00
[ready]: b*a              // multiply row vector 'b' by
matrix 'a'
(10^0) X
row 0 =
  3.00   3.00   3.00
 [ready]: a*b             // an invalid multiplication
>>>error = 4 ---incompatible dimensions
[ready]: b*4              // multiply each element of 'b' by
4
row 0 =
  4.00   4.00   4.00
```

*. *Array Element Multiplication*

Syntax: a *. b

Description: The operator symbol *. is used to perform the multiplication of corresponding elements between two arrays. The multiplication is implemented as:

$$c_{j,i} = a_{j,i} \cdot b_{j,i} \quad \text{,for all j, i;}$$

where j and i are row and column indices respectively. An error message will be generated if 'a' and 'b' do not have the same dimensions.

Example: The following MSHELL statement will multiply corresponding elements in array 'a' and 'b', and store the result in 'c',

```
[ready]: a = 1::2::3      // creates a row vector
[ready]: a                    // print array
row 0 =
  1.00   2.00   3.00
[ready]: b = a
[ready]: c = a*.b         // mult. corresponding elements
[ready]: c                    // print results
row 0 =
  1.00   4.00   9.00
```

\\ *Continuation Line delimiter*

Description: The symbol **** tells the MSHELL interpreter that the present statement will be continued on the next physical line. Continuation lines can only be used within MSHELL script files (***.msf** or ***.msh**)

Example: The following 2 lines of code illustrate the use of the continuation line delimiter within a script function. The result of the expression is 21.

```
x = 4*5 + \\
1
```

// *Single Line Comment Delimiter*

Description: The symbol **//** is used to specify that a single line comments will follow. All characters that follow this symbol within a given physical line, will be ignored.

Example: The following illustrates the use of the single line delimiter.

```
[ready]: pi = 4*atan(1.)  // this text will be ignored
```

/ *Array Division by Scalar*

Syntax: a **/** b

Description: The operator symbol **/** is used to perform the division of each element of an array by a scalar. An error message will be generated if 'b' is not a scalar, i.e. a 1x1 array. The actual division is implemented as:

$c_{j,i} = a_{j,i} / b$ for all j, i;

where j and i are row and column indices respectively.

/. *Element Array Division*

Syntax: a **/.** b

Description: The operator symbol **/.** is used to perform the division of corresponding elements between two arrays. An error message will be generated if 'a' and 'b' do not have the same dimensions. The actual division is implemented as:

$c_{j,i} = a_{j,i} / b_{j,i}$ for all j, i;

where j and i are row and column indices respectively.

Example: The following MSHELL statement will divide the elements in array 'a' by the elements in array 'b', and attempt to divide elements in array 'a' by a scalar.

```
[ready]: a = 1::3::5      // generates a row vector
[ready]: b = a
[ready]: a/.b             // divide corresponding elements
row 0 =
  1.00   1.00   1.00
[ready]: a/.4             // since 'x' is not 1x1 an error
occurs
>>>error = 4 ---incompatible dimensions
```

^ *Raise Array Elements to a Power*

Syntax: a **^** b

Description: The operator symbol **^** or caret is used to raise the elements of an array to a given constant power, or the power specified by the corresponding elements of another array. The above statement will raise the elements in array 'a' to the power specified by the elements in array 'b', where 'a' and 'b' have the same dimensions. For 'a' and 'b' input arrays, we have the following:

If 'a' and 'b' have the same dimension, the actual operation is implemented as:

$c_{j,i} = a_{j,i}^{b_{j,i}}$ for all j, i;

If 'b' is a scalar (i.e. a 1 x 1 array) and 'a' is not, the operation is implemented as,

$c_{j,i} = a_{j,i}^{b_{0,0}}$ for all j, i;

If ‘a’ is a scalar (i.e. a 1x1 array) and ‘b’ is not, the operation is implemented as,

$$c_{j,i} = a_{0,0}^{b_{j,i}} \quad \text{for all j, i;}$$

where j and i are row and column indices respectively. Note that in each case the result ‘c’ is an array; only in the case of both ‘a’ and ‘b’ scalars, will ‘c’ be a scalar.

Example:

```
[ready]: x = (1::3)#(4::5)    // creates a 2 by 2 matrix
[ready]: x                    // print 'x'
row 0 =
  1.00   3.00
row 1 =
  4.00   5.00
[ready]: 2^x              // raise 2 to 'x'
row 0 =
  2.00   8.00
row 1 =
 16.00  32.00
[ready]: x^0.5                // raises 'x' to 0.5
row 0 =
  1.00   1.73
row 1 =
  2.00   2.24
[ready]: x^x              // raises corresponding elements
row 0 =
  1.00  27.00
row 1 =
256.00 ###.##
[ready]: M_format = "00000.00"
[ready]: x^x
row 0 =
    1.00    27.00
row 1 =
  256.00  3125.00
```

' *Matrix Transpose*

Syntax: a'

Description: The operator symbol **'** is used to generate the transpose of an array. Mathematically, this operation is implemented as:

$$c_{row=i,col=j} = a_{row=j,col=i} \quad \text{for all j, i;}$$

where j and i are row and column indices respectively in ‘a’.

Example: These MSHELL statements assigns the transpose of ‘a’ to ‘c’.

```
[ready]: data = randg(3,3)    // generate a 3x3 random array
[ready]: data             // print 'data'
row 0 =
   -0.18     0.10     1.36
row 1 =
    0.61     0.57     1.23
row 2 =
   -1.10     1.55    -0.58
[ready]: data'                //  transpose of 'data'
row 0 =
   -0.18     0.61    -1.10
row 1 =
    0.10     0.57     1.55
row 2 =
    1.36     1.23    -0.58
```

:: *Concatenate Arrays or Strings*

Syntax: a **::** b

Description: Given two arrays with the same number of rows, this operation will append the corresponding rows of one array to the other array. Likewise, given two strings (which can be considered one row arrays or row vectors) this operation will append the two strings.

Example: For the arrays 'a' and 'b', the following MSHELL statement will append the rows of 'a' to the rows of 'b', and store the result in 'c'.

```
[ready]: c = a::b
```

The number of columns in 'c' is equal the sum of the number of columns in 'a' and 'b'. The following MSHELL commands will assign to the variable 'y' the concatenation of two arrays.

```
[ready]: x = 1::2             // creates a row vector
[ready]: y = x::x         // concatenates 2 vectors
[ready]: x
row 0 =
   1.00   2.00
[ready]: y
row 0 =
   1.00   2.00   1.00   2.00
```

The following MSHELL commands will assign to the string variable '$str1' the concatenation of two strings.

```
[ready]: $str = "SSS"     // creates a string
[ready]: $str1 = $str::" last name"   //concatenates another
string
[ready]: $str1
SSS last name
```

Column Augmentation

Syntax: a # b

Description: Given two arrays with the same number of columns, this operation will append the corresponding columns of one array to the other array.

Example: The following MSHELL statement will append the columns of 'b' to the columns of 'a', and store the result in 'c'.

```
[ready]: c = a#b
```

The number of rows in 'c' equals the sum of the number of rows in 'a' and 'b'. The following MSHELL commands illustrate the above,

```
[ready]: a = (0,3,1)'     // creates a column vector
[ready]: b = ones(2,1)        // creates a 2-d identity
column vector
[ready]: c = a#b          // augments a with b and assigns
result to c
```

= Assignment

Syntax: c **=** a

Description: This operator symbol, **=** , is used to assign the output of an MSHELL expression to a variable. MSHELL will not allow you to assign an array expression to an already defined string variable, or vice versa.

Example: The following MSHELL statement will assign the sum of two constants to the newly defined variable 'c'.

```
[ready]: c = 4 + sqrt(3.333)
```

If the variable 'c' has already been defined its content will be changed, otherwise it will be created.

: Array Interval Delimiter

Syntax: x(**:** ,n)

Description: Used to denote an interval of rows or columns.

Example: The following illustrates the use of the array interval delimiter:

To extract column number 3 of an array 'x', type:

```
[ready]: x(:,3);
```

To extract row number 2 in 'x' and assign it to 'y', type:

```
[ready]: y = x(2,:);
```

To extract the sub array in 'x' defined by rows 3 to 5 and columns 2 to 10, and copy to 'y' type:

```
[ready]: y= x( 3:5 , 2:10 )
```

To insert the array 'y' into the array 'x' starting at row 8 and column 0 type:

```
[ready]: x( 8: , 0: ) = y
```

To extract an image block starting at row 9 and column 20 type

```
[ready]: x(9:, 20: )
```

$string String Access Control

Syntax: **$*string*(rs:re,cs:ce) or $string(cs :ce) or …**

Description: The notation for accessing the content of a string variable is discussed in page 22, the string variable section.

Logical Relational Operators

<	Less Than Operator
>	Greater Than Operator
<=	Less Than or Equal To Operator
>=	Greater Than or Equal To Operator
==	Equivalent To Operator
!=	Inequivalent To Operator
&&	Logical AND Operator
||	Logical OR Operator

Syntax: *expression1* **OPERATOR** *expression2*

Description: These operators are used to perform logical checks on certain desired expressions.

Example: Notice the use of separating parenthesis to establish precedence in the example below:

```
if ( (I==4.8)  &&  (P>3) ){
        x=y;
}
```

- A -

abs Absolute Value

Syntax: **abs**(a) or a.**abs**

Description: Returns the absolute value of each element in the array.

Example: The following MSHELL statement will compute the absolute values for each element of 'a'.

```
[ready]: a=1::-2::3::-4   // stores these values on variable
a
[ready]: abs(a)
row 0 =
   1.00    2.00    3.00   4.00
```

acos Inverse Cosine

Syntax: **acos**(a) or a.**acosor**

Description: Returns the inverse cosine of each array element. The output is in radians. The actual mathematical expression computed is given by,

$c_{j,i} = \text{acos}(a_{j,i})$ for all j, i;

where j and i are row and column indices respectively.

Example: The following MSHELL statement will compute the inverse cosine of 'a', and store the result in 'c'.

```
[ready]: c = acos(a)
```

addmenuitem Adds a User Defined Menu Item

Syntax: **addmenuitem** "myscript.msh"

Description: Use this command to add the capability of invoking user defined script files from within the graphical user interface, where "myscript.msh" is a valid script file.

Example: The following line will add the 'mdemo.msh' file to the menu item list under User

```
[ready]: addmenuitem   "mdemo.msh"
```

all_local all_local

Description: keyword used by 'save' function. See 'save' for details.

allstr allstr

Description: keyword used by 'save' function. See 'save' for details.

all Allows for "all" Variables to be included

Syntax: free **all**

Description: Use this command to access all variables at one time. It can be used with show all, to free all variables, or with free all, to free all variables.

Example: The following line will free all variables from memory

```
[ready]: free all
```

allocate_array Allocate an array and set its values

Syntax: allocate_array(nrows, ncols, value)

Description: Allocate a new ARRAY variable with the specified dimension and set all its elements to the specified value.

Example:

```
[ready]: x = allocate_array(10, 10, 0)       // same as "x =
zeros(10, 10)"
[ready]: x = allocate_array(10, 10, 1)       // same as "x =
ones(10, 10)"
```

See also: ones, zeros

---

alternateRGB describe array as an RGB volume

Syntax: alternateRGB(x)

Description: Use this command to let MSHELL know that when accessing the user specified variable, say 'x', to use an alternate volumetric representation. In essence this commands sets the internal variable attributes 'x.m_m_=3' and 'x.m_voldim' to match a band sequential RGB representation. Once the 'alternateRGB()' command is executed any view command will make use of the alternate representation. To read the voldim information use: x.m_nrows3d, x.ncols3d, and x.m_nband3d.

Example: see alternateBSQ( ) example.

---

alternateBSQ describe array as a BSQ volume

Syntax: alternateBSQ(x,Nrows,Ncols,Nbands)

Description: Use this command to let MSHELL know that when accessing the user specified variable, say 'x', to use an alternate volumetric representation. In essence this commands sets the internal variable attributes 'x.m_voltype=1' , i.e. a band sequential (BSQ) representaionand. It also sets the 'x.m_voldim' to
x.m_voldim=Nrows,Ncols,Nbands. If the user provided alternate representation is not possible it will return an error message.
Once the 'alternateBSQ()' command is executed any view command will make use of the alternate representation. To read the voldim information use: x.m_nrows3d, x.m_ncols3d, and x.m_nband3d.

Example:

```
free all
// load an image and creatre a BSQ representation
// using shifted version of the image

xp = reada("eqohare.chr","char");        // load image
y = shiftc(xp,1,1);              // shift by one pixel
z = shiftc(xp,15,15);        // shift by 15 pixels
layers = xp#y#z;          // create BSQ cube
show layers

if(1){
          alternateBSQ(layers,xp.nrows,xp.ncols,3)   //
arrange
          layers.text = "Band sequential image with 3 bands.
\n Each band are shifted versions of the same image."
          view layers                // request view
}
if(1){
          myColorImage = layers
          alternateRGB(myColorImage);
          view myColorImage
}
if(1){
          roi = wdef(0,0,xp.nrows,xp.ncols)
          green_extracted = layers(roi|2);               //
read a band
          view green_extracted
}
if(1){
          // now let us look at a complex variable
          myComplexImage = complex(xp,z);
          myComplexImage .text ="has real an imaginary part
=> red=real, green=imaginary"
          view myComplexImage            // display complex
variable
}

// the following is another way to construct and access a
cube
nbands = 3
layers2 = zeros(  nbands*xp.nrows, xp.ncols)
layers2( : |  0 )     = xp
layers2( : |  1 )     = yp
layers2( : |  2 )     = zp
green2        = layers2( | ,  2)


print "done"
```

and *Bitwise AND operator*

Syntax: and(x, y)

Description: Perform bitwise AND operator treating input arguments as unsigned 64 bits integers. Input arguments can be also real ARRAYs; in this case, the bitwise AND operator is performed on corresponding elements.

Example:

```
[ready]: x = 3; y = 5;
[ready]: and(x, y)
   1.00
```

aoi *Active Region of Interest*

Syntax: image.**aoi**

Description: **This command lists the pixel values within a previously selected region of interest.**

***Note**: The region of interest must be of conservative size considering the requirements to list all values within the region.*

asin *Inverse Sine*

Syntax: **asin**(a) or a.**asin**

Description: Compute the inverse sine of each array element. The output is in radians. The actual mathematical expression computed is given by,

$c_{j,i} = \mathrm{a}\sin(a_{j,i})$ for all j, i;

where j and i are row and column indices respectively.

Example: The following MSHELL statement will compute the inverse sine of 'a', and store the result in 'c'.

```
[ready]: c = asin(a);
```

atan *Inverse Tangent*

Syntax: **atan**(a) or a.**atan**

Description: Compute the inverse tangent of each array element. The output is in radians. The actual mathematical expression computed is given by,

$c_{j,i} = \text{atan}(a_{j,i})$ for all j, i;

where j and i are row and column indices respectively. The returned values are between **-π/2** and **+π/2**.

Example: The following MSHELL statement will compute the inverse tangent of 'a' and store the result in 'c'.

```
[ready]: c = atan(x);
```

atan2 Inverse Tangent

Syntax: **atan2**(y,x)

Description: Computes the inverse tangent of each array element. The output is in radians. Returns the arc tangent of y/x (in the range **-π** to **+π**); **atan2** produces correct results even when the resulting angle is near **-π/2** or **+π/2** (i.e. x near 0).

Note that the input values must be in the range of **-1** to **+1**, otherwise incorrect results will be generated, also note that if both x and y are set to 0, **atan2**(y,x) is set equal to **1**.

- B -

blackw *Blackman-Harris Window*

Syntax: **blackw**(n,m)

Description: Generates a 2-D (4 coefficient) Blackman-Harris Window. The n^{th} element of the 1-D Blackman-Harris Window is defined as,

$$w_n = c_0 - c_1 \cdot \cos(\frac{2\pi}{N} n) + c_2 \cdot \cos(\frac{2\pi}{N} 2n) + c_3 \cdot \cos(\frac{2\pi}{N} 3n)$$

where,

$$c_0 = 0.35875,\ c_1 = 0.48829,\ c_2 = 0.14128,\ \text{and}\ c_3 = 0.01168.$$

This window has side lobes that are -92dB below the main lobe.

Example: The following generates a 256 x 256 Blackman-Harris Window.

```
[ready]: y = blackw(256,256)
[ready]: [YSpectrum] = spectrum[y]
[ready]: view Yspectrum
[ready]: roi = wdef(256-16,256-16,32,32)
[ready]: plot3d(YSpectrum(roi))
```

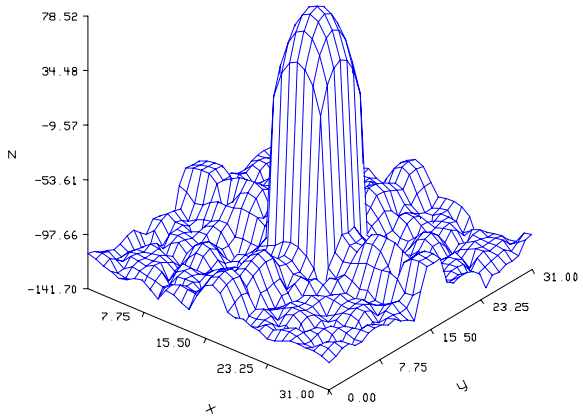


Figure 2

blinterp Bi-linear Interpolation

Syntax: **blinterp**(f,z)

Description: Blinterp will bi-linearly interpolate between data points located in a two dimensional square grid, e.g. an image. Where 'f' is a 2-dimensional real data array containing the input image. The data points in 'f' correspond to points in a square grid with an assumed interpixel distance of 1 in the row or column axis and 'z' is a complex data array containing the vertices at which the input image is to be bi-linearly interpolated. Specifically, the real part of 'z' contains the fractional column positions and the imaginary part contains the fractional row positions at which the input image is to be interpolated. Note that for every point that extrapolation is attempted the result is set to the system variable 'M_no_data'. The value of this variable can be modified by the MSHELL user.

Example: The following example generates a simple test pattern image over a 4x4 grid. The image is then bilinearly interpolated at the row coordinate 2.2 and column coordinate 1.2.

```
[ready]: f = hammiw(4,4)  // test image
[ready]: f
row 0 =
  0.01   0.04   0.08   0.04
row 1 =
  0.04   0.29   0.54   0.29
row 2 =
  0.08   0.54   1.00   0.54
row 3 =
  0.04   0.29   0.54   0.29
[ready]: x = complex( 2.2, 1.2)
[ready]: blinterp(f,x)
        0.573856
```

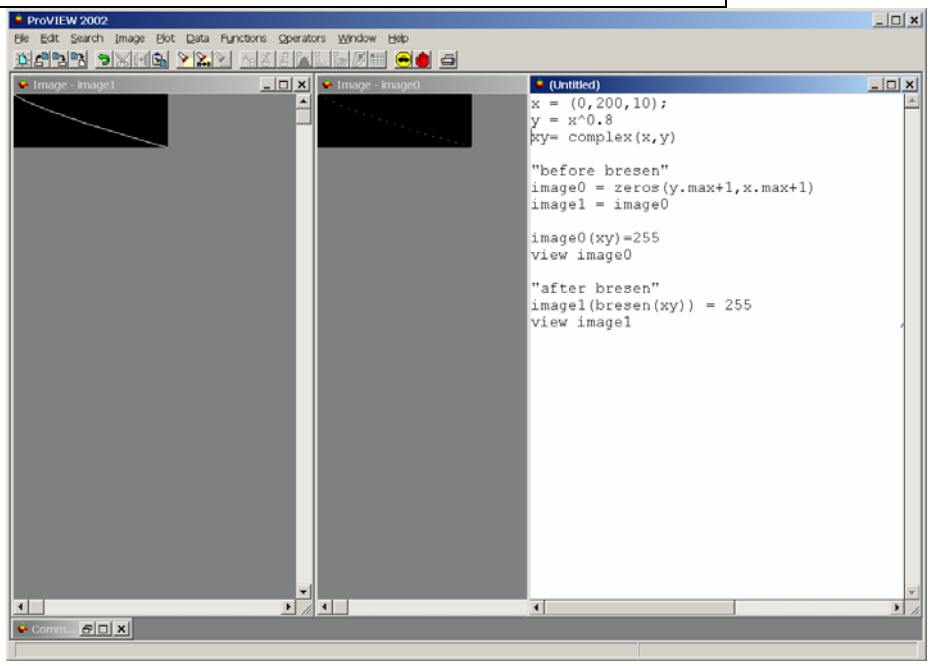
bresen Compute Line Segment Points

Syntax: **bresen**(z) or z.**bresen**

Description: Given the coordinates of two or more end points in the plane, this function computes the points along the line segment between the points using the Bressenham's Line Drawing Algorithm. This function is particularly useful to define a region of interest make out of line segments. The argument 'z' is a complex row data vector in which the nearest integer of the real part of z is used as the column positions, and the nearest integer of the complex part of z as the row positions.

Example: The following uses **bresen** to connect to points in a list.

```
x = (0,200,10);
y = x^0.8 // generate points on a parabola
xy= complex(x,y)

"before bresen"
image = zeros(y.max+1,x.max+1)

image(xy)=255
view image

"after bresen"
image = image*0
image(bresen(xy)) = 100
view image
```



bthresh Binary Threshold

Syntax: **bthresh**(a,tval)

Description: Given an input array, ‘a’, and a threshold value, ‘tval’, this function returns a clipped version of the input array. That is, every element in the input array less than the real threshold value is set to zero, and every element greater or equal than the threshold value is set to 1.

Example: The following illustrates the use of bthresh.

```
[ready]: z = (0,5,1,4);
[ready]: z
row 0 =
  0.00   1.00   2.00   3.00   4.00   5.00
row 1 =
  0.00   1.00   2.00   3.00   4.00   5.00
row 2 =
  0.00   1.00   2.00   3.00   4.00   5.00
row 3 =
  0.00   1.00   2.00   3.00   4.00   5.00
[ready]: bthresh(z,3)
row 0 =
  0.00   0.00   0.00   1.00   1.00   1.00
row 1 =
  0.00   0.00   0.00   1.00   1.00   1.00
row 2 =
  0.00   0.00   0.00   1.00   1.00   1.00
row 3 =
  0.00   0.00   0.00   1.00   1.00   1.00
```

- C -

callDLL Calls a DLL for execution

Syntax: **callDLL(**$dllname,$function,returned_paramters**)**;

Description: This function is used to call a DLL for execution. It must have been previously loaded with loadDLL. The DLL must not return any more than 5 values. The syntax is DLL filename, function name, returns(comma delimited). See section "Dynamic Link Libraries (DLL)" for details.

Example: The following line calls the delaunay.dll for execution after having been loaded with loadDLL. There is only a 'z' variable being returned by this dll; therefore, one must return zeros for all unused placeholders.

```
[ready]: callDLL("delaunay.dll","delaunay",z,0,0,0,0)
```

ceil Find the Ceiling of an Array

Syntax: **ceil**(array);

Description: This function is used to find the ceiling of an array. The output has the same dimensions as the original array. The output contains the ceiling for each element

Example: [ready]: x = ceil(1.2::3.4)

row 0 =

2.0 4.00

3.0

centroid computes centroid

Syntax: **centroid(x,1)**

Description: computes the centroid based on the amplitude values in an image. This function uses all pixels in the image for the computation.

closef *Close a File*

Syntax: **closef**(unit);

Description: This function is used to close a disk file previously opened using the **openf** function. It's argument, 'unit' is the integer file number assigned when **openf** was initially invoked. Failure to close a file could result in a future error when doing disk i/o.

See Also: **openf**

cmirror *Mirrors an Image Column Wise*

Syntax: **cmirror**(a) or a.**cmirror**

Description: Mirrors the columns of the input array or image, 'a'.

Example: The following illustrates the operation on the array 'aa'.

```
[ready]: aa = (1::2::3)#(4::5::6)#(7::8::9)
[ready]: aa
row 0 =
  1.00   2.00   3.00
row 1 =
  4.00   5.00   6.00
row 2 =
  7.00   8.00   9.00
[ready]: cmirror(aa)      // print mirror array of aa
row 0 =
  3.00   2.00   1.00
row 1 =
  6.00   5.00   4.00
row 2 =
  9.00   8.00   7.00
```

cmplxoverlap *2-D range index*

Syntax: **cmplxoverlap(x,roi)**

Description: UNDER CONSTRUCTION

colplot Plots a Row from an Array

Syntax: **colplot[N]**(array,column#)

colplot[N](array, -1) ← for interactive plot

Description: Used to plot a particular column from an array. See also rowplot().

complex Creates a Complex Array

Syntax: **complex**(x,y)

Description: Given 'x' and 'y' as real arrays with equal dimension, this function constructs an array, 'z', of the form z = x+iy, where i = sqrt(-1).

Example: The following is a simple application of the function.

```
[ready]: complex(0::3,4::3)
(10^0) X
row 0 =
  0.00+  4.00i   3.00+  3.00i
```

conj Array Conjugate

Syntax: **conj**(a) or a.**conj**

Description: This function returns the complex conjugate of each element in the input array 'a' , i.e. it changes the sign of the complex part.

Example: For example, if x = 4+3i, then **conj**(x) returns 4-3i.

contour Generates a contour plot

Syntax: **contour**[*#*](z,zlevels) or **contour**[*#*](z,yaxis,xaxis,zlevels)

Description: This function generates a contour plot of the image or sub image region selected where: 'z' is a real array, with at least 2 rows and 2 columns; 'zlevels' is the row vector with the contour levels that will be used; and '#' is the Plot Screen Number.

Instead of 'zlevels' you can use the following instruction to generate 11 different levels between the maximum and minimum value of z:

```
z.min+(z.max-z.min)*(0,1,0.1)
```

.The '#' parameter in the contour function is an optional integer number (between 0 and 255) that selects the plot screen where the plot will be placed. If an integer number from 0 to 255 is provided in this field the generated plot can be indexed from then on by that number. For example, if x is an array, then

```
contour3(z,0.5::0.75)
```

will plot a contour plot of array 'z' on plot screen number 3, using for contour levels 0.5 and 0.75. The second way of calling contour allows to specify the values to use in for the x-axis and y-axis. If later on you want to free that screen type:

```
free plot3
```

See Also: "M_" for a complete list of system variables which affect the plot related functions.

Example: The following MSHELL instructions generate a 32 x 32 hamming function, which is then stored in 'x', and then used to generate a contour plot of 'x'.

```
[ready]: M_xlabel = "row index"; M_ylabel = "column index"
[ready]: x = hammiw(32,32)
[ready]: contour10(x,x.min+(x.max-x.min)*(0,1,0.3))
[ready]: contour(x, (0,31,1)*10,  (0,31,1)*0.1, 0.5)
```

---

convol Discrete Convolution

Syntax: **convol**(k,a)

Description: This function performs the discrete convolution of a given array, 'a', with a kernel array, 'k'. The implementations used in **convol** and **convolt** are computationally efficient for small kernel sizes. However, for large kernels, an **FFT** implementation of the convolution should be considered.

Given an image and the right kernel this function can be used to change the spatial resolution in the image.

Given that array 'a' has dimensions (N x M),

$$a \rightarrow \begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,M-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,M-1} \\ \vdots & \vdots & \cdots & \vdots \\ a_{N-1,0} & a_{N-1,1} & \cdots & a_{N-1,M-1} \end{pmatrix}$$

it is assumed that 'a' is zero for any index outside of the implicit range specified above. It is required that the kernel, 'k', be of odd dimension, say (2P+1 x 2Q+1), where P and Q are non-negative integers satisfying:

$$2P+1 \le N, \text{ and } \quad 2Q+1 \le M.$$

The input kernel samples are assumed to map into a row range of (-P to P) and a column range of (-Q to Q) ,

$$k \rightarrow \begin{pmatrix} k_{-P,-Q} & \cdots & k_{-P,+Q} \\ \vdots & \cdots & \vdots \\ k_{+P,-Q} & \cdots & k_{+P,+Q} \end{pmatrix}$$

The result of the convolution is the array 'g',

$$g \rightarrow \begin{pmatrix} g_{-P,-Q} & \cdots & g_{N-P,M+Q} \\ \vdots & \cdots & \vdots \\ g_{N+P-1,-Q} & \cdots & g_{N+P,M+Q} \end{pmatrix}$$

whose elements $g_{j,i}$ are given by,

$$g_{j,i} = \sum_{p=-P}^{P} \sum_{q=-Q}^{Q} (k_{p,q} \cdot a_{j-p,i-q}),$$

By construction, indices outside of the specified range of values in the above 'g' matrix are zero. Note that the output of **convol** will have dimensions of (N+2P, M+2Q).

See Also: **"convolt"** below

Example: **"convolt"** below

---

convolt *Truncated Discrete Convolution*

Syntax: **convolt**(k,a)

Description: This function is similar to convol; in many instances you may only be interested in determining how the elements in the range of 'a' are affected by the convolution operation. This function truncates the convolution results by only evaluating 'g' over the range of 'a' , i.e., a row range of (0 to N-1) and a column range of (0 to M-1).

The implementations used in **convol** and **convolt** are computationally efficient for small kernel sizes. However, for large kernels, an **FFT** implementation of the convolution should be considered.

Example: For arrays 'a' and 'k' as define, **convol** and **convolt** are given as:

```
[ready]: a = (1,3,1)
[ready]: k = (3,1,1)
[ready]: convol(a,k)      // example of convol
row 0 =
  3.00   8.00  14.00   8.00   3.00
[ready]: convolt(a,k)     // example of convolt
row 0 =
  8.00  14.00   8.00
```

convtoi Convert Row Vector to Image

Syntax: **convtoi**(a,ncols)

Description: Converts a 1-D row array , 'a', to a 2-D array where you specify the column dimension, 'ncols'. The resulting number of rows must be an integer. That is, if the row vector had dimensions (1 x M), and the you are questing a column dimension of N, the output array will have dimensions of ((M/N) x N),i.e. N must be a factor of M.

Example: The following converts the row array , 'a', to a 2-D array.

```
[ready]: a = 1::2::1::2   // create row vector
[ready]: convtoi(a,2)     // convert row vector to 2-col.
array
row 0 =
  1.00   2.00
row 1 =
  1.00   2.00
```

convtov Convert Image to Row Vector

Syntax: **convtov**(a) or a.**convtov**

Description: Converts the 2-D input array to a 1-D array containing only one row. That is, if the input array, 'a', had dimensions (N x M), the output array will have dimensions of (1 x (N * M)).

Example: The following converts the 2-D array, 'a', to a row array.

```
[ready]: a = (1,2,1,2)        // creates a 2x2 matrix
[ready]: a
row 0 =
  1.00   2.00
row 1 =
  1.00   2.00
[ready]: convtov(a)       // converts matrix to row vector
row 0 =
  1.00   2.00   1.00   2.00
```

cos *Cosine*

Syntax: **cos**(a) or a.**cos**

Description: Returns the cosine of each array element. The input is expected to be in radians.

Example: The following MSHELL statement will compute the cosine of 'a' divided by 2, and store the result in 'c'.

```
[ready]: c = cos(a/2)
```

cosh *Hyperbolic Cosine*

Syntax: **cosh**(a) or a.**cosh**

Description: Computes the hyperbolic cosine of each array element. The hyperbolic cosine of x is evaluated as:

$$\cosh(x) = \frac{e^{x} + e^{-x}}{2}$$

covm *Covariance Matrix Estimation*

Syntax: **covm**(a) or a.**covm**

Description: Computes an unbiased estimate of the covariance matrix established by the column vectors in 'a'. The actual estimation is done using the following equation,

$$\frac{1}{I-1} \cdot \sum_{i=0}^{I-1} (a_i - a_c)(a_i - a_c)^t$$

where, a_i is the i^{th} column in 'a', and a_c = **sumc**(a) is the column vector resulting by averaging each row in 'a'.

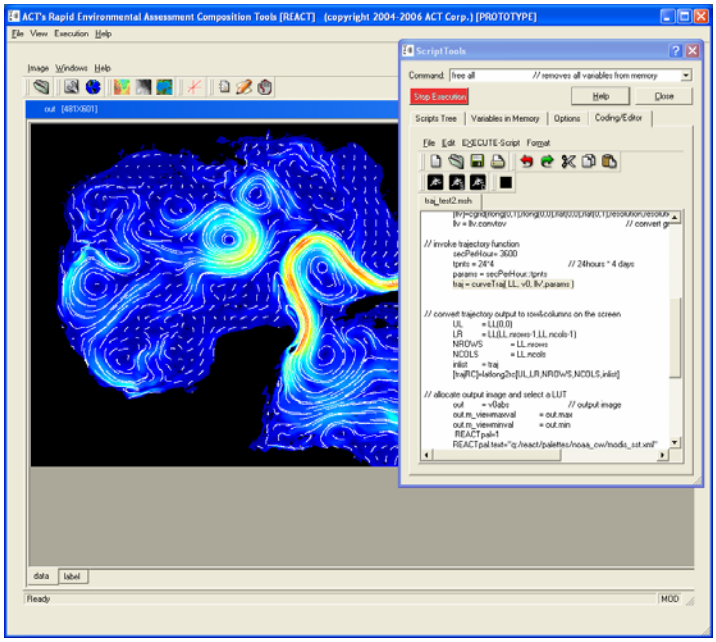
See Also: Fukunaga, K., "Introduction to Statistical Pattern Recognition", for a detailed definition.

curveTraj *particle Trajectory flow builder*

Syntax: **traj = curveTraj(LL, v0, posStart,params)**

Description: Computes trajectory flow of particles by solving Lagrangian equations with marching and iteration. Note: The original implementation of this algorithm was in FORTRAN and provided by Dr. Ko at NRL/SSC. The inputs to this routine are:

- **LL** – latitude and longitude planes (complex array). Values are in degrees decimal
- **v0** – complex array with user provided velocity (cm/sec) components in U (east positive) and V (north positive), i..e. v0 = complex(U,V). This array has the same dimensions as 'LL'
- **posStart** – complex column vector (Nx1) with longitude and latitude positions to use as starting point. Input is in degrees decimal.
- **params** – row vector with interval value of seconds to use, and number of points to use in the tracing, e.g. params=deltaT::num_points.
- **Traj** – return complex array with trajectories computed. The dimension of this array is (posStart.ncols X num_points). Each row is a single particle trajectory. The first point in the row corresponds to the starting input position.



```
// read the lat&long+U&V components (in raw form)
          $fname=M_proviewdir::"/../react_data/nasa_ca/nrl/ne
tcdf/u3d_2004110900.nc"
          [u,LL,depth,time] = ncom4d_nc_extract[$fname]

          $fname=M_proviewdir::"/../react_data/nasa_ca/nrl/ne
tcdf/v3d_2004110900.nc"
          [v,LL,depth,time] = ncom4d_nc_extract[$fname]

          LL = rmirror(LL)

          roi = wdef(0,0,LL.nrows,LL.ncols)
          v0 = rmirror( complex(u(roi|0),v(roi|0)) )     //
will hold the inputs u_x&v_y components

          groi = eqindex(v0.real,-99)        // set no
velocity over ground
          v0(groi)=complex(0,0)              // set to 0
those points for which we do not have valid data values
          v0abs = v0.abs
          view v0abs

// select starting trajectory points to use
          resolution = 0.5
          rlong      = maxmin(LL.real)
          rlat       = maxmin(LL.imag)
          [llv]=cgrid[rlong(0,1),rlong(0,0),rlat(0,0),rlat(0,
1),resolution,resolution] // create a grid of interesting
points
          llv = llv.convtov                  // convert grid
to a vector

// invoke trajectory function
          secPerHour= 3600
          tpnts = 24*4           // 24hours * 4 days
          params = secPerHour::tpnts
          traj = curveTraj( LL, v0, llv',params )


// convert trajectory output to row&columns on the screen
          UL  = LL(0,0)
          LR  = LL(LL.nrows-1,LL.ncols-1)
          NROWS   = LL.nrows
          NCOLS       = LL.ncols
          inlist  = traj
          [trajRC]=latlong2rc[UL,LR,NROWS,NCOLS,inlist]

// allocate output image and select a LUT
          out = v0abs         // output image
          out.m_viewmaxval    = out.max
          out.m_viewminval    = out.min
           REACTpal=1
```

```
          REACTpal.text="q:/react/palettes/noaa_cw/modis_sst.
xml"
          [REACTrgb]=get_noaa_palette[REACTpal]
          if( ncols(out.LUT   )==0){
           out.LUT    = (0,255,1,3)
          }
          if( ncols(out.LUTdef)==0){
          out.LUTdef = out.LUT
          }
          out.LUT=REACTrgb

          out(trajRC.convtov)=100;

          lastpos = trajRC(:,trajRC.ncols-1)
          out(lastpos')=255

           out(groi)  = 0;
          view out
          wtile
```

- D -

dbclose Closes access to an external database

Syntax: **dbclose(**$database**)**

Description: Use this command to close the alerady opened database.

Example: see dbrsgetrows example

dbrsrowcount Returns the result set count

Syntax: **dbrsrowcount()**

Description: This function works after a dbconnect and dbsqlexec. After a query this function when invoked will return the number of rows in the result set.

Example: see dbrsgetrows example

dbrsgetrows Returns the result set as a string data.

Syntax: **dbrsgetrows(**srow,erow**)**

dbrsgetrows(srow,erow,$fname**)**

dbrsgetrows($colnames,srow,erow**)**

dbrsgetrows($colnames,srow,erow,$fname**)**

where,

srow – start row number

erow – end row number

$fname – output file name (string)

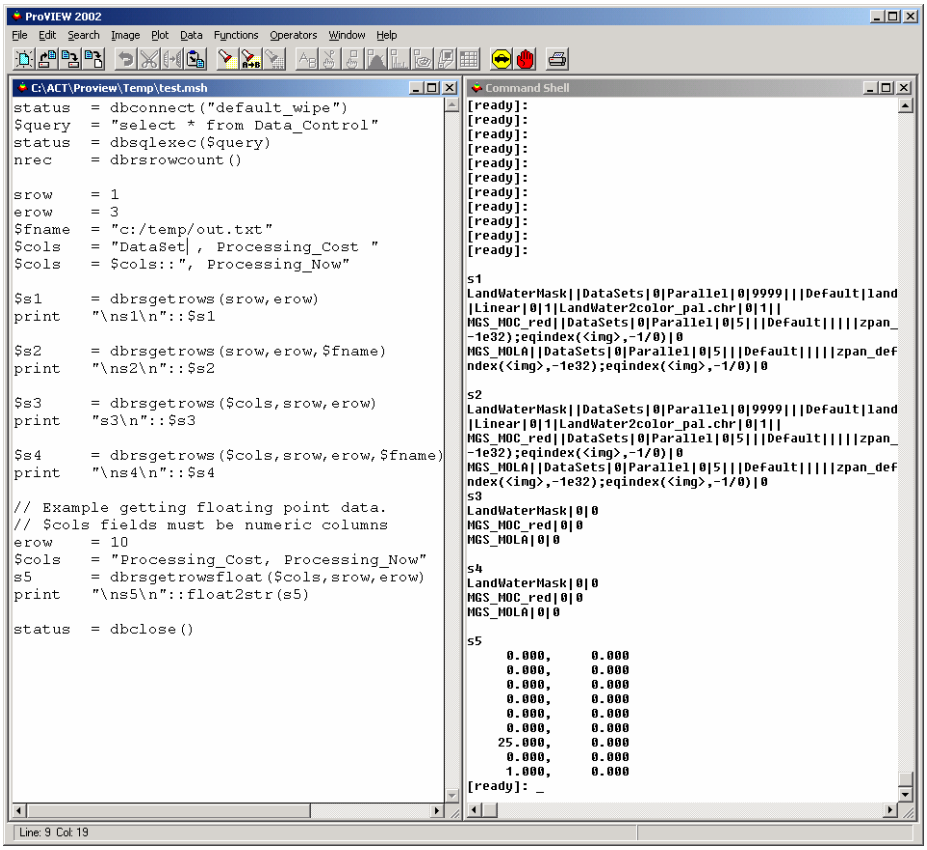
$colnames – list of requested field names

Description: This function works after a dbconnect and dbsqlexec. After a query this function when invoked will return the result set data as a string. The columns are separated by a | and rows are

separated by new lines. This function may be called multiple times on the same result set. . There are different ways to extract the results of the query using this function

Example: The following example shows most of the dbrs… function being utilized on a simple test case.

```
status    = dbconnect("default_wipe")
$query    = "select * from Data_Control"
status    = dbsqlexec($query)
nrec      = dbrsrowcount()

srow      = 1
erow      = 3
$fname  = "c:/temp/out.txt"
$cols   = "DataSet , Processing_Cost "
$cols     = $cols::", Processing_Now"

$s1       = dbrsgetrows(srow,erow)
print     "\ns1\n"::$s1

$s2       = dbrsgetrows(srow,erow,$fname)
print     "\ns2\n"::$s2

$s3       = dbrsgetrows($cols,srow,erow)
print     "s3\n"::$s3

$s4       = dbrsgetrows($cols,srow,erow,$fname)
print     "\ns4\n"::$s4

// Example getting floating point data.
// $cols fields must be numeric columns
erow      = 10
$cols     = "Processing_Cost, Processing_Now"
s5        = dbrsgetrowsfloat($cols,srow,erow)
print     "\ns5\n"::float2str(s5)

status    = dbclose()
```



dbconnect Connects to database for access .

Syntax: **dbconnect** ($database)

Description: Use this command to connect to an external datbase. The string passed is the name which has been previouly defined for the database in the ODBC administration tool. Returns a value of **zero** if it is successful.

Example: see dbrsgetrows example

dbrsgetrowsfloat Returns the result set as an array.

Syntax: **dbrsgetrowsfloat(**srow,erow**)**

dbrsgetrowsfloat(srow,erow,$fname**)**

dbrsgetrowsfloat($colnames,srow,erow**)**

dbrsgetrowsfloat($colnames,srow,erow,$fname**)**

where,

srow – start row number

erow – end row number

$fname – output file name (string)

$colnames – list of requested field names

Description: This function works after a dbconnect and dbsqlexec. After a query this function when invoked will return the result set data as an array. The first input argument $column_name is a comma separated list of column names to be retrieved. This function may be called multiple times on the same result set. If your are retrieving a large number of numeric related records this function executes significantly faster than 'dbrsgetrows'.

Example: see dbrsgetrows example

---

dbsqlexec Sends a query to the database

Syntax: **dbsqlexec($query)**

Description: This function executes a query on the connected server, it is used after a dbconnect . Returns a value of zero if it is successful.

Example: see dbrsgetrows example

---

dbfopen Returns a handle to the opened dbf file

Syntax: **dbfopen($dbffile)**

Description: this function returns a handle to the opened dbf file if it succeeds, otherwise it returns –1. You can open up to 5 dbf files at a time.
Note: In MS-Windows a dbf file **can also** be accessed using the Visual FoxPro ODBC driver. All the functions of the type dbf___ are for giving access to the dbf file. These functions do not provide SQL query capability.

Example:

```
$dbffile = "../../react_data/global/place_find/country.dbf"

hdbf      = dbfopen($dbffile)     // get handle

fldcnt    = dbfgetfieldcnt(hdbf)  // get number of
fields/record and name of the fields
print     "number of records="::int2str(fldcnt)
print     "\nThe field names are:\n"
print     fldcnt.text                //names of the fields
print     "\n"


nrec      = dbfgetrecordcnt(hdbf) // get number of records
print     "number of records are: " :: int2str( nrec)::"\n"

$rec = dbfgetrecords(hdbf,-1,-1)  // read all the records
print "showing the content of the 1st 3 records..\n"
$rec(0:2,:)

status    = dbfclose(hdbf)           // close the handle
```

dbfclose This function closes the dbfile

Syntax: **dbfclose(hdbf)**

Description: this function closes the dbffile. It makes the handle available for a subsequent call to dbfopen.

Example: see example in dbfopen( )

dbfgetfieldcnt returns the # of fields in the dbf file.

Syntax: dbfgetfieldcnt(hdbf)

Description: this function returns the number of fields present in the dbf file. It also returns the names of the fields in the text attribute of the returned array variable.

Example: see example in dbfopen( )

dbfgetrecordcnt returns the # of records in the file

Syntax: **dbfgetrecordcnt(hdbf)**

Description: This function returns the number of records in the dbf file .

Example: see example in dbfopen( )

---

dbfgetrecords read the records from the dbf file.

Syntax: **dbfgetrecords(hdbf,-1,-1)**
dbfgetrecords(hdbf,recVec,fieldVec)

Description: This function is used to read the records from the dbf file. The input parameters are the
hdbf - dbffile handle,
recVec - an arrary variable containing the record indices (zero based) you wish to extract. If set to -1, then all records are extracted.
fieldVec - an array variable containing the field indices (zero based) you want to extract. If set to -1, then all fields are extracted.

The output is a string containing the field values, separated by '|', for the specified rows or all the rows .

Example: see example in dbfopen( )

dbrsgetcolnamesReturns column names in a result set

Syntax: **dbrsgetcolnames** ()

Description: Returns a comma separated list of the column names in a result set. To be called after a SELECT query call to sqlexec(). Any expressions in the SELECT statement may be returned as an empty column name (DB driver dependent), unless the AS operator is used. The column name should match the AS identifier properly.

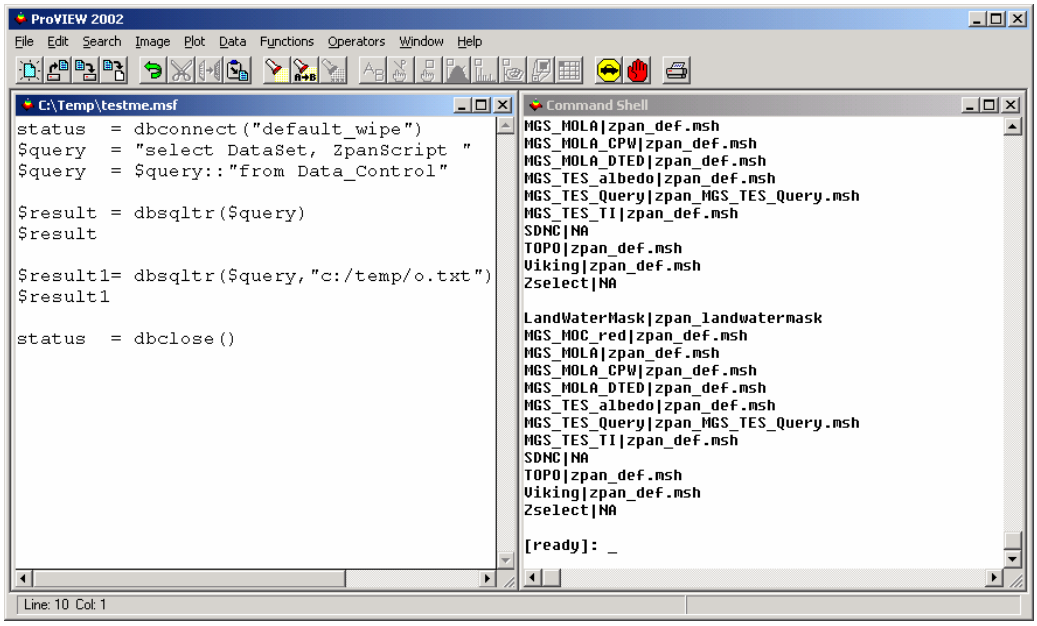
dbsqltr *Transacts with an external database*

Syntax: **dbsqltr(**$query,<$filename>**)**

Description: Use this command to post the passed string as the next query for the previously connected database. The output of the query, if any, is automatically retrieved. The records are extracted and string formatted. The fields are delimited by the character '|'. The output is assigned to a string variable if the optional filename parameter is not specified; otherwise, the output is saved to the named file.

Example:

```
status  = dbconnect("default_wipe")
$query  = "select DataSet, ZpanScript "
$query  = $query::"from Data_Control"

$result = dbsqltr($query)
$result

$result1= dbsqltr($query,"c:/temp/o.txt")
$result1

status  = dbclose()
```



dct8x8 Discrete Cosine Transform (8x8)

Syntax: **dct8x8**(a)

Description: Computes the discrete cosine transform of each 8 x 8 block within the specified array "a".

decimate Signal Decimation

Syntax: **decimate**(a,rowskip,columnskip)

Description: Extracts a sub-sampled version of the input signal, where: 'a' is the input array, 'rowskip' and 'columnskip' are the number of rows and columns, respectively, to be skipped in each direction.

Example: A 512 x 512 input image, 'a', can be decimated to a 128 x 128 image, 'b', using the following MSHELL statement,

```
[ready]: b = decimate(a,4,4);
```

dirlist Returns the list of subdirectories/files

Syntax: **dirlist($path ,$options) or sys_dirlist($path,$optons)**

Description: This function returns the list of the subdirectories or files of a given directory. The first input argument is a string, containing the directory name. The second input argument is one of the following "d" , "s" or, "p" .
When "d" is used the function will return only the list of subdirectories. Otherwise only the files are listed. When "s" is used the function will return the list of files along with their size. When "p" is used the function will list the path of the files .
Note: this function uses '|' as the delimiter.

Example:

```
// to print directories, their sizes, and the full path
$out = dirlist("c:\\temp\\*.*", "dsp" )

// to print the directory listing only.  This will not print
files
$x = dirlist("F:/crism_pds_archive/trdr/TRDR/*","d")
```

diskfreespace returns disk info

Syntax: **diskfreespace("c:/temp")**

Description: This function returns an array variable containing three values. The first value is the total number of bytes in the disk, the second value is the total number of free bytes in the disk, and the third value is the total number of free bytes per caller (this can be used if disk quotas are setup per user). The input argument is a string containing a path (UNC are allowed).

Example:

```
[ready]: freespace = diskfreespace("e:\\")
[ready]: freespace = diskfreespace(\\\\hyper\\cdrive)
```

dt2mjd Returns the modified Julian Date

Syntax: **dt2mjd(Year::Month::Day::Hour::Minute::Second)**

Description: This function returns the modified Julian date, the input argument is year, month, day, hour, minute, and seconds (includes fractions of seconds).

Note: In the single precision version of MSHELL, using mjd2dt after dt2mjd is accurate to within minutes. In the double precision version it is accurate to mseconds.

Example: **see tdiff()**

DDEExec Sends a command to the server app.

Syntax: **DDEExec(chan,cmd)**

Description: This command is used to send a command to the server application once the DDE conversation is established . chan is the channel number returned by DDEInit, and cmd is a string you want the server to execute. DDExec will return 0 if the server accepted the command, 1 otherwise.

Example: See User's Manual page 54.

DDEInit initialize DDE server application

Syntax: **chan = DDEInit(“Application Name”,”Topic”)**

Description: This command is used to start a conversation on a particular topic with the server application. “Application Name” is the name of the application you want to communicate with (Ex. MSAccess), and “Topic” is the name of the particular topic. This command returns a channel number associated with the conversation you are requesting or -1 if the operation fails.

Example: See User’s Manual page 54.

DDEPoke POKE DDE server

Syntax: **DDEPoke(chan,item,val)**

Description: This command is used to request the server application to accept an unsolicited data item value. Chan is the channel number returned by DDEInit, item is a string that identifies the data item you wish to send a value a value for, and val is a string containing the value. DDEPoke will return 0 if the server accepted the data item value, 1 otherwise.

Example: See User’s Manual page 54.

DDEReqS Requests value from DDE server

Syntax: **$var = DDEReqS(chan,$item)**

Description: This command is used to request the server application to provide the value of a data item. $var is a MSHELL string variable, chan is the channel number returned by DDEInit, and item is a string that identifies the data item you are requesting the value of.

Example: See User’s Manual page 54.

DDEReqV Requests value from DDE server

Syntax: **DDEReqV(chan,$item)**

Description: This command is used to request the server application to provide the value of a data item. Syntax : var = DDEReqV(chan, $item). ‘var’ is a MSHELL array variable, chan is the channel number returned by DDEInit, and item is a string that identifies the data item you are requesting the value of.

Example: See User’s Manual page 54.

DDETerm close a DDE conversation

Syntax: **DDETerm(chan)**

Description: This command terminates a conversation. DDETerm(chan) where chan is the number returned by DDEInit.

Example: See User's Manual page 54.

dworld draws world coastline map

Syntax: **dworld**(llrange,outcntl)

or

dworld(xyrange,outcntl**)**

Description: There are two modes of usage for this function. Both modes use the world data file pointed by the system variable 'M_wdb'. *Under your MSHELL installation directory, see the subdirectoy 'wdb' for a list of possible files to use.*

F**irst usage mode -** In this mode , 'dworld', generates an image corresponding to the coastline map of the world for the selected range of latitudes and longitudes (in degrees). In this mode the output map is generated faster and in simple cylindrical type of projection.

llrange – a 1x4 row vector that determines the area of interest in latitude and longitude, e.g.
llrange = latmin::latmax::longmin::longmax

outcntl – a 1x2 row vector that control the number of rows and columns in the output image, e.g. coldim = 300::600 will. It is optional to add a third parameter to 'outcntl'. This third parameter can be used to read every 'gap' pixel from the input world data file. For example if the output image is 300 rows and 400 columns and you desire to read every 25th point from the input world data file then, coldim = 300::400::25.
The maximum gap value recommended is 45. For global coverage large gap sizes can be used, but for small map areas of interest smaller values in the gap size generate a more accurate coast line output. The default value of gap is internally adaptive depending on the coverage size of the selected area.

Second usage mode – In this mode 'dworld' generates the output in any of the supported cartographic projections accessible via the function 'Geo2Cart()'.

xyrange - a 1x4 row vector, corresponding to the desired region in the cartographic domain (default units are meters), .e.g..

xyrange= ymin::ymax::xmin::xmax

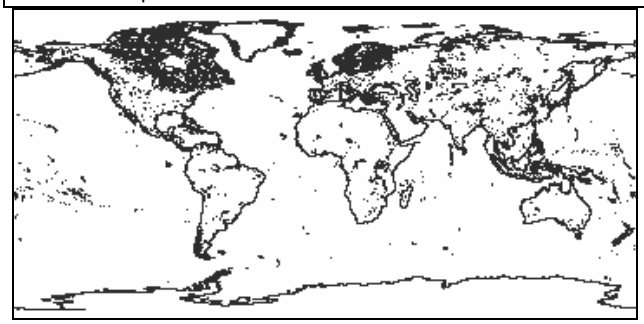
xyrange.text –must be set to the desired **proj4** string (see examples and Geo2Cart() description). If 'dworld' detects any content in xyrange.text it will assume that the user is interested in generating its output as a cartographic map.

outcntl – same as above definition

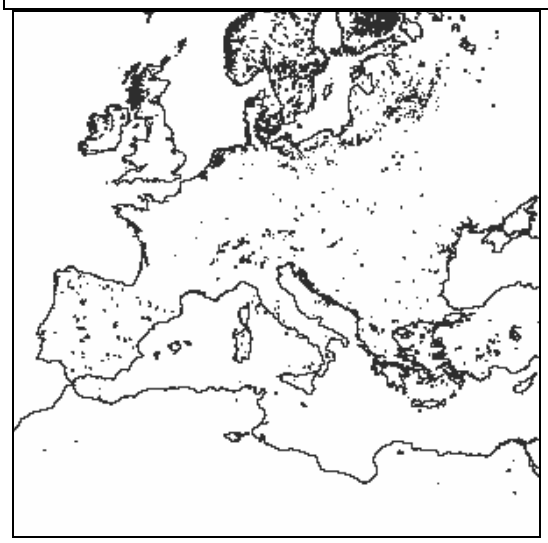
See Also: wdbmap.msf, Ml_geo2cart.msf , Geo2Cart(), Map_Image.msf, Map_Images.msf

Example:

```
M_wdb = "C:/ACT/Proview/wdb/coasts.dat"  // use coast line
weqc = dworld(-90::90::-180::180,180::360)
weqc = 255-weqc   // negate the image to force white
background
view255 weqc
```

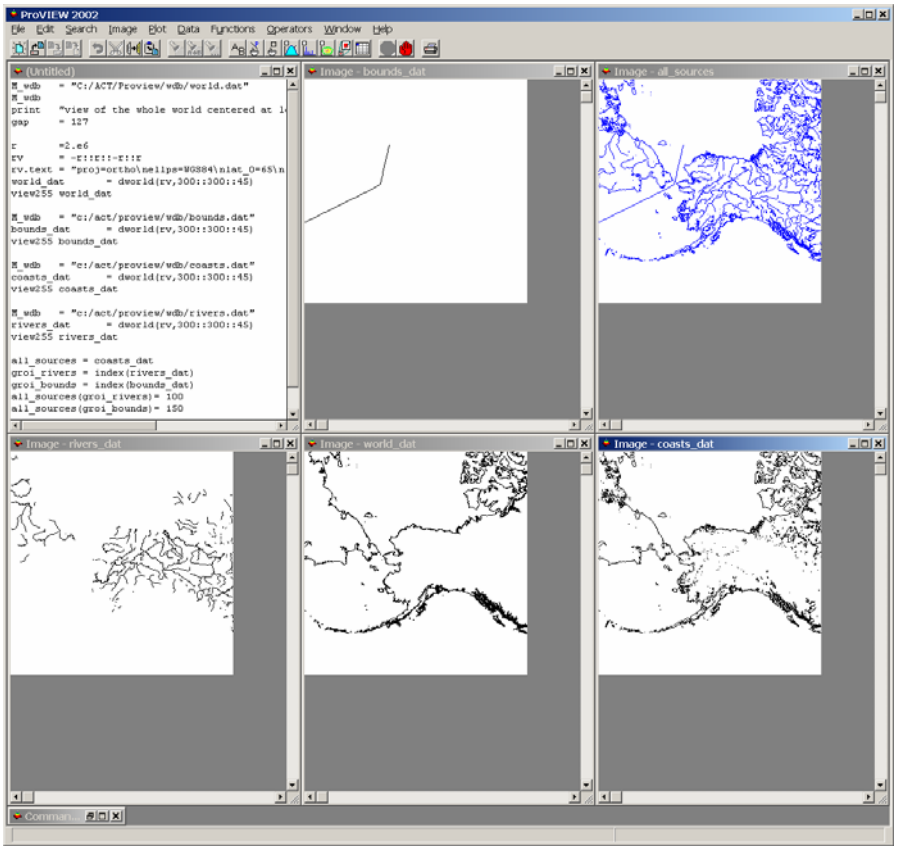


```
r             = 2.0e6
range         = -r::r::-r::r
range.text = "proj=ortho \nellps=WGS84 \nlat_0=45
\nlon_0=12\nno_defs"
range.text = smodify(range.text," ","");  // remove spaces
wview         = dworld(range,300::300)
wview         = 255-wview
view255   wview
```



The following example shows sample output from all the input files that can be accessed via the M_wdb variable. The input data use here comes from the CIA vector shoreline publicly available.

```
M_wdb     = "C:/ACT/Proview/wdb/world.dat"
M_wdb
print     "view of the whole world centered at longitude 0\n"
gap       = 127

r         =2.e6
rv        = -r::r::-r::r
rv.text   = "proj=ortho\nellps=WGS84\nlat_0=65\nlon_0=-
155\nno_defs" // center on Alaska
world_dat = dworld(rv,300::300::45)
view255 world_dat

M_wdb     = "c:/act/proview/wdb/bounds.dat"
bounds_dat    = dworld(rv,300::300::45)
view255 bounds_dat

M_wdb     = "c:/act/proview/wdb/coasts.dat"
coasts_dat    = dworld(rv,300::300::45)
view255 coasts_dat

M_wdb     = "c:/act/proview/wdb/rivers.dat"
rivers_dat    = dworld(rv,300::300::45)
view255 rivers_dat

all_sources = coasts_dat
groi_rivers = index(rivers_dat)
groi_bounds = index(bounds_dat)
all_sources(groi_rivers)= 100
all_sources(groi_bounds)= 150
view all_sources
```



Additional examples can be obtained by executing the following code.

```
M_wdb     = "c:/act/proview/wdb/bounds.dat"
M_wdb     = "C:/ACT/Proview/wdb/world.dat"
M_wdb
print     "view of the whole world centered at longitude 0\n"
gap       = 45
orows     = 180
ocols     = 360
weqc      = dworld(-90::90::-180::180,orows::ocols::gap)
LLw       = 255-weqc
view255   LLw     // equi-distance cylindrical view of the
world

M_wdb     = "C:/ACT/Proview/wdb/coasts.dat"
M_wdb
print     "view of the whole world centered at longitude 0\n"
weqc      = dworld(-90::90::-180::180,orows::ocols::gap)
LLc0      = 255-weqc
view255   LLc0    // eqc view of the world centered at long=0

print     "view of the whole world centered at longitude 0,
midpoint between 0::360\n"
weqc      = dworld(-90::90::0::360,orows::ocols::gap)
LLc180    = 255-weqc
view255   LLc180

r         = 10e6
rv        = -r::r::-2*r::2*r
rv.text   = "proj=sinu\nellps=WGS84\nlat_0=-
90\nlon_0=0\nno_defs"
wview     = dworld(rv,orows::ocols::gap)
sinu      = 255-wview
view255 sinu

M_wdb     = "C:/ACT/Proview/wdb/world.dat"
M_wdb
r         = 10e6
rv        = -r::r::-2*r::2*r
rv.text   = "proj=moll\nellps=WGS84\nlat_0=-
90\nlon_0=0\nno_defs"
wview     = dworld(rv,orows::ocols::gap)
mollweid= 255-wview
view255 mollweid

r         =3.e6
rv        = -r::r::-r::r
rv.text   = "proj=ortho\nellps=WGS84\nlat_0=65\nlon_0=-
155\nno_defs"
// centered on Alaska
wview     = dworld(rv,300::300::45)
ortho_AK= 255-wview
view255 ortho_AK

r         =2.5e6
```

```
rv        = -r::r::-r::r
rv.text   =
"proj=ortho\nellps=WGS84\nlat_0=40\nlon_0=13\nno_defs"
// centered on ITALY
wview     = dworld(rv,300::300::45)
ortho_IT= 255-wview
view255 ortho_IT
```

- E-

END *Ends Execution of a Script*

Syntax: **END**

Description: Used within a script to end the execution. It is to be placed at the end of the script when to be completed. No error condition or message is generated when this function is called.

eqindex *Equality Index*

Syntax: **eqindex**(a,b)

Description: Finds the locations of all the elements in an input array which equal a constant value, where 'a' is the input array and 'b' is the constant scalar quantity. This function returns a (1 x M) complex array, where M is the number of points equal to the specified value and whose array elements contain the coordinates of each point encoded as follows: the real part contains the column index, and the imaginary. part contains row index of the point. If no elements are matched, a NULL array is returned.

Example: Cases with matches and without are illustrated below.

```
[ready]: x = (0,5,1)
[ready]: eqindex(x,3)
3 + 0i
[ready]: y = eqindex(x,10)
--- NULL ARRAY ---
[ready]: y = eqindex(x,10)
[ready]: y.ncols
    0
```

See Also: finite_index(),gt_index, leindex(),ltindex(), geindex(), nan_index(), …The following illustrates the use of some of these functions

```
z=0           // zero value
show z
minf = -1/0       // - infinity
show minf
nan = -1/0+1/0    // generates a Not a Number
show nan
uno = 1
show uno

v = z::minf::nan::uno // construct a vector
v

print "index(v)       // index to elements that are not
zero\n"
index(v)

print "\n finite_index(v) // index to elements that are
finite\n"
finite_index(v)

print "\n eqindex(v,z)    //index to elements equal to 0 \n"
eqindex(v,w)

print "\n eqindex(v,minf) //index to elements equal to -
infinity \n"
eqindex(v,minf)

print "\n eqindex(v,nan)  //index to elements equal to NaN
\n"
eqindex(v,nan)

print "\n nan_index(v)    // index to Not-a-Number
elements\n"
nan_index(v)

print "\n ltindex(v,uno)  // index to elements less than
one\n"
ltindex(v,uno)

print "\n rindex(v,-1/0,1000) // index to all elements
between minus infinity and 1000\n"
rindex(v,-1/0,1000)

print "\n geindex(v,1)    // index to all elements >=1 \n"
geindex(v,1)
```

eqindexS Equality Index String

Syntax: **eqindexS**($a,$b)

Description: Finds the locations of all the elements in an input string which equal a defined substring, where '$a' is the input string and '$b' is the substring. This function returns a (1 x M) complex array, where M is the number of points equal to the specified value and whose array elements contain the coordinates of each string position encoded as follows: the real part contains the column index, and the imaginary. part contains row index of the location. If no elements are matched, the value **-1** is returned.

See Also: The functions **strContains**($a,$b), **strStartWith**($a,$b), and **strEndsWith**($a,$b) may be more convenient to use, depending on the user needs.

Example:

```
[ready]:
[ready]: $string = "This is an example to demonstrate the use of eqindex$."
[ready]: x = eqindex$($string,"de")
[ready]: x

(10^0) X
row 0 =
 22.00-  0.00i
row 1 =
 49.00-  0.00i

[ready]: _
```

---

evaltext *Evaluates a String*

Syntax: **evaltext** $str

Description: This function allows you to send a string to the MSHELL interpreter for execution. You can use this function to create variable names within a script file.

Example: The following lines of code illustrate the application.

```
       i = 1
$str = "x"::int2str(i)::"= hammiw(32,32)*255"
evaltext $str
view x1
num = 5
       evaltext "$string = \"This is a very good manual.\""
       $string
```

---

exit *Exit MSHELL*

Syntax: **exit**

Description: This command will exit the MSHELL Interpreter and confirm if you want to exit the MSHELL environment and, if so, return you to the host operating system.

exp *Inverse-Natural Logarithm*

Syntax: **exp**(a) or a.**exp**

Description: Computes the inverse natural logarithm of each array element in 'a', i.e., raise e to the power of each array element. The actual mathematical expression computed is given by,

$c_{j,i} = e^{a_{j,i}}$ for all j, i

where j and i are row and column indices respectively, and $e = \ln(1)$.

Example: The following MSHELL statement will compute the inverse-logarithm of 'a' and store the result in 'c',

```
[ready]: c = exp(a)
```

expand *Environment variables expansion*

Syntax: **expand(** $string **)**

Description: Expand any system environment variable contained in '$string' to its value. Environment variables must be specifed in the form '$(VAR)'.

Example:

```
[ready]: expand("Qt dir is $(QTDIR)")
         Qt dir is D:\Qt\3.3.7
```

- F -

fft *1-D Fast Fourier Transform*

Syntax: **fft**(a) or a.**fft**

Description: Computes the one dimensional Fourier transform of the rows of the input array, 'a'. The input as well as the output of this function may be a complex array. Note that the row dimension of the input array must be a power of two.

Example: If 'a' is an input array with dimensions of 64 x 64, then the one dimensional Fourier transform of each row can be computed as, **fft**(a).

fft2 *2-D Fast Fourier Transform*

Syntax: **fft2**(a) or a.**fft2**

Description: Computes the two dimensional Fourier transform of the input array. The input as well as the output of this function may be a complex array. It is expected that the dimensions of the input array are a power of two.

Example: If 'x' is an input array with dimensions of 64 x 64, then its power spectrum in dB can be estimated using the following construction,

```
Power_spectrum = 20 * real(log10(fft2(x)))
```

See Also: spectrum.msf; Appendix B External Function

fileinfo *Returns detailed information of a file*

Syntax: **fileinfo**($fname)

Description: Returns the size of an existing file in bytes and the date and time created. If the file does not exist it returns **-1**.

filesize Returns the size of a file

Syntax: **filesize**($fname)

Description: Returns the size of an existing file in bytes. If the file does not exist it returns **-1**.

findfiles Locate Files in Directory Structure

Syntax: **findfiles**($path,$type)

Description: This function can be used to find all files starting at a given directory level, '$path', and satisfying a matching criteria, '$type'. The function returns a string with all the files that satisfy the matching criteria. If '$path' contains any environment variables in the form '$(VAR)', those are expanded to their values (see "expand" command).

Example: The following returns all files with extension "chr" in c:\proview.

```
[ready]: $str = findfiles("c:\proview","*.chr")
[ready]: $str
c:\proview\IMAGES\EQOHARE.CHR
c:\proview\IMAGES\MANDEL.CHR
c:\proview\IMAGES\MSHELL.CHR
c:\proview\IMAGES\RM000.CHR
```

finite_index Index of all non-infinite elements

Syntax: **finite_index**(a)

fits_open_file Open a fits file

Syntax: **handle = fits_open_file**($file_name, “r”)

Description: Opens a FITS file, and returns a handle for use with reading and writing functions. The second parameter specifies reading (“r”) or writing (“w”)

Example:

```
[ready]:handle = fits_open_file("C:/NXEROS_0001/XRS00053.FIT
","fits")
[ready]: handle
1
```

fits_get_num_hdus Returns the number of HDUs

Syntax: **num_of_hdus = fits_get_num_hdus(handle)**

Description: Returns the number of HDU (Header and Data Unit**)** from a previously opened fits file. Note that an HDU may consist entirely of a header with no data records.

Example:

```
[ready]: handle = fits_open_file("C:/NXEROS_0001/XRS00053.FIT
","fits")
[ready]: num_of_hdus = fits_get_num_hdus(handle)
[ready]: num_of_hdus
2
```

fits_movabs_hdu Moves to a specified hdu number

Syntax: **hdu_type = fits_movabs_hdu(handle,hdu_number)**

Description: Moves to the specified hdu_number, so hdu_number is less or equal to num_of_hdus. HDU type:

IMAGE_HDU 0 /*Primary Array or IMAGE HDU*/

ASCII_TBL 1/*ASCII table HDU */

BINARY_TBL 2 /*Binary table HDU */

Example:

```
[ready]: handle = fits_open_file("C:/NXEROS_0001/XRS00053.FIT
","fits")
[ready]: num_of_hdus = fits_get_num_hdus(handle)  //
num_of_hdus =  2
[ready]: HDU_type = fits_movabs_hdu(handle,1)
[ready]: HDU_type
0
```

fits_get_hdu_type - Returns type number of current HDU

Syntax: **type_num = fits_get_hdu_type(handle)**

Description: Returns the type number of current HDU. The possible types are:

IMAGE_HDU 0 /*Primary Array or IMAGE HDU*/

ASCII_TBL 1 /*ASCII table HDU */

BINARY_TBL 2 /*Binary table HDU */

Example:

```
[ready]: handle = fits_open_file("C:/NXEROS_0001/XRS00053.FIT
","fits") [ready]: type_num = fits_get_hdu_type(handle)
[ready]: type_num    // note that
0
```

fits_get_num_keywords - Returns # of existing keywords

Syntax: **num_keywords = fits_get_num_keywords(handle)**

Description: Returns the number of existing keywords (not counting END keyword).

Example:

```
[ready]: handle = fits_open_file("C:/NXEROS_0001/XRS00053.FIT
","fits")
[ready]: num_keywords = fits_get_num_keywords(x)
[ready]: num_keywords
738
```

fits_read_keyn_name Returns keyname

Syntax: **$keyname = fits_read_keyn_name(handle,keynum)**

Description: Returns a string $keyname for the key number keynum . Note that keynum can not exceed the number of existing keywords.

Example:

```
[ready]: handle = fits_open_file("C:/NXEROS_0001/XRS00053.FIT
","fits")
[ready]: num_keywords = fits_get_num_keywords(handle)
[ready]: $keyname = fits_read_keyn_name(handle,keynum)
//keynum <= 738
[ready]: $keyname   // from previous example num_keywords is
738
TTYPE173            // for keynum = 647
```

fits_read_keyn_value Returns a keyvalue

Syntax: **$keyvalue = fits_read_keyn_value(handle,keynum)**

Description: Returns a string $keyvalue for the key number keynum. Note that keynum can not exceed the number of existing keywords.

Example:

```
[ready]: handle = fits_open_file("C:/NXEROS_0001/XRS00053.FIT
","fits")
[ready]: num_keywords = fits_get_num_keywords(handle)
[ready]: $keyvalue = fits_read_keyn_value(handle,keynum)
//keynum <= 738
[ready]: $keyvalue         // from previous example
num_keywords is 738
'unf_level_1_safing_flg'   // for keynum = 647
```

fits_read_keyn_comment Returns keyword comment

Syntax: **$keycomment = fits_read_keyn_comment(handle,keynum)**

Description: Returns keyword comment $keycomment for the key number keynum. Note that keynum can not exceed the number of existing keywords..

Example:

```
[ready]: handle = fits_open_file("C:/NXEROS_0001/XRS00053.FIT
","fits")
[ready]: num_keywords = fits_get_num_keywords(handle)
[ready]:$keycomment=fits_read_keyn_comment(handle,keynum)
//keynum<= 738
[ready]: $keycomment   //from previous example num_keywords
is 738
label for field 173    // for keynum = 647
```

fits_read_ keyword_value Returns Key value

Syntax: **$keyvalue = fits_read_keyword_value(handle,$keyname)**

Description: Returns a string $keyvalue for an existing name $keyname . **Example:**

```
[ready]: handle = fits_open_file("C:/NXEROS_0001/XRS00053.FIT
","fits")
[ready]: $keyname = fits_read_keyn_name(handle,keynum)
//keynum = 647
[ready]: $keyvalue = fits_read_keyword_value(x,$keyname)
[ready]: $keyvalue
'unf_level_1_safing_flg'
```

fits_read_keyword_comment Returns Keyword comment

Syntax: **$keycomment= fits_read_keyword_comment(handle,$keyname)**

Description: Returns keyword comment string for $keyname.

Example: The following MSHELL statement will compute the….

```
[ready]: handle = fits_open_file("C:/NXEROS_0001/XRS00053.FIT
","fits")
[ready]: $keyname=fits_read_keyword_name(handle,keynum)
//keynum = 647
[ready]: $keycomment =
fits_read_keyword_comment(handle,$keyname)
[ready]: $keycomment
label for field 173
```

fits_get_size Returns the # of rows and columns

Syntax: **size = fits_get_size(handle)**

Description: Returns the number of rows and columns in the current FITS file. Size(0,0) = number if rows and Size(0,1) = number of columns.

Example:

```
[ready]: handle = fits_open_file("C:/NXEROS_0001/XRS00053.FIT
","fits")
[ready]: size   = fits_get_size(handle)
[ready]: size
    124.0     195.0
```

fits_get_colnum Returns the column # of a fits file

Syntax: **colinfo = fits_get_colnum(handle,$template_ name)**

Description: Returns a vector of columns, and a string list in colinfo.text of colnames (newline separated) if more than one column match the pattern. $template_name is the exact name of the column to be searched for, or it may contain a wild card character(*,?,or #), or it may contain the integer number of the desired column (with the first column = 1).

Example:

```
[ready]: handle = fits_open_file("C:/NXEROS_0001/XRS00053.FIT
","fits")
[ready]: colinfo = fits_get_colnum(x,"met")
[ready]: colinfo
1
```

fits_get_coltype Returns the column type

Syntax: **coltype = fits_get_coltype(handle,colnum)**

Description: Returns the column type number, repeat count (vector columns). Returns 1x3 vector

Coltype(0,0) = column type number

Coltype(0,1) = repeat count

Coltype(0,2) = width

Example:

```
[ready]: handle = fits_open_file("C:/NXEROS_0001/XRS00053.FIT
","fits")
[ready]: coltype = fits_get_coltype(handle,1)  // first
column
[ready]:coltype
     41.0       1.0       4.0
```

fits_read_col_values Reads column values

Syntax: **coldata = fits_read_col_values(handle,col,firstrow,firselem,melem) or**

coldata = fits_read_col_values(handle,col,firstrow,num_rows) or

coldata = fits_read_col_values(handle,col)

Description: Reads the column values starting with firstrow, firstelem, and continuing for nelem (crossing into new rows if needed). The coldata output will always be a 1xN vector using this form. Even if the column is a vector.

Example:

```
[ready]: handle = fits_open_file("C:/NXEROS_0001/XRS00053.FIT
","fits")
[ready]:coldata=fits_read_col_values(handle,col,firstrow,firs
elem,melem)
```

fits_read_col_str Reads column strings

Syntax: **$coldstr =** fits_read_col_str(handle,col,firstrow,firselem,melem) or

$coldstr = fits_read_col_str(handle,col,firstrow,num_rows) or

$coldstr = **fits_read_col_str(handle,col)**

Description: Reads the column values converted to strings starting with firstrow, firstelem, and continuing for nelem (crossing into new rows if needed). The coldata output will always be a 1xN vector using this form. Even if the column is a vector.

.

Example:

```
[ready]: handle = fits_open_file("C:/NXEROS_0001/XRS00053.FIT
","fits")
[ready]: colstr =
fits_read_col_str(handle,col,firstrow,firselem,melem)
```

fits_close_file closes a fits file

Syntax: **status = fits_close_file(handle)**

Description: Closes a previously open fits file.

Example:

```
[ready]: status = fits_close_file(handle)  // handle = 1 (
from the previous example )
[ready]: status
0
```

FLEVEL internal keyword to MSHELL

Syntax:

Description: internal keyword used by MSHELL

float2str Convert Array to a formatted String

Syntax: **float2str(a)** or **float2str(a,$format)**

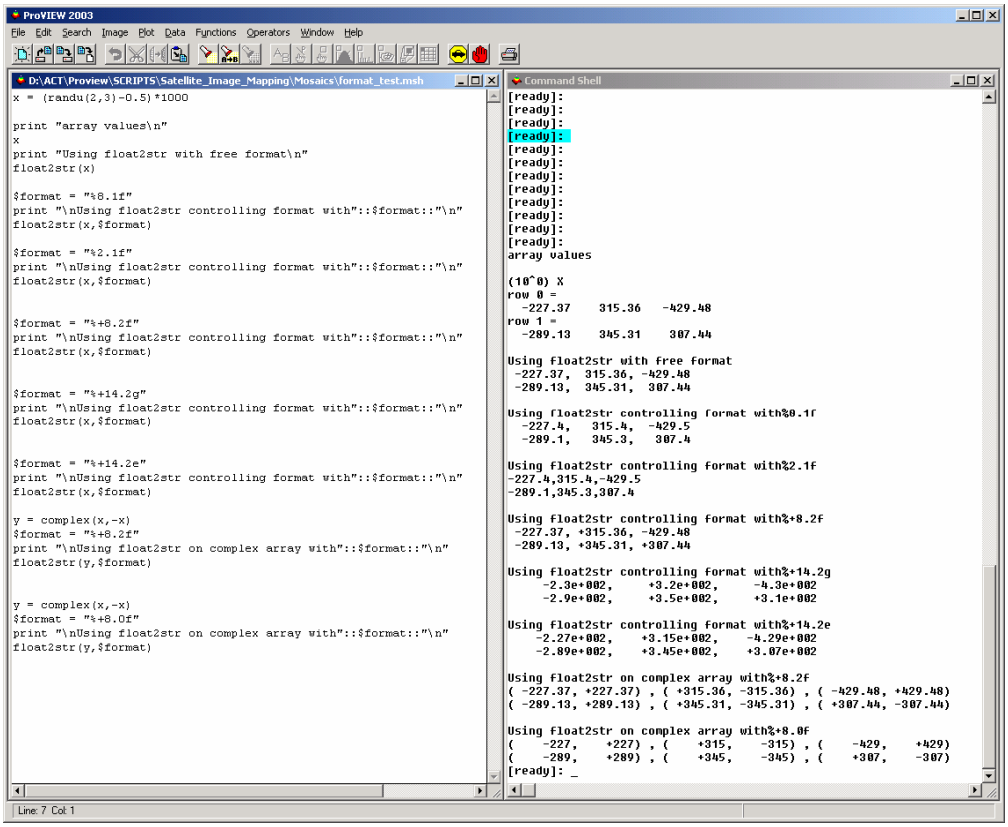
Description: Converts the input array 'a' to a string representation. The user can control the format of each element by providing a controlling string, i.e. $format, which follows the standard C

language approach. If the input is complex, the data will be displayed in groups of of (x,y) or (real,imaginary).

Example

The following gives usage examples follow by the actual output.

```
x = (randu(2,3)-0.5)*1000
print "array values\n"
x
print "Using float2str with free format\n"
float2str(x)

$format = "%8.1f"
print "\nUsing float2str controlling format
with"::$format::"\n"
float2str(x,$format)

$format = "%2.1f"
print "\nUsing float2str controlling format
with"::$format::"\n"
float2str(x,$format)

$format = "%+8.2f"
print "\nUsing float2str controlling format
with"::$format::"\n"
float2str(x,$format)

$format = "%+14.2g"
print "\nUsing float2str controlling format
with"::$format::"\n"
float2str(x,$format)

$format = "%+14.2e"
print "\nUsing float2str controlling format
with"::$format::"\n"
float2str(x,$format)

y = complex(x,-x)
$format = "%+8.2f"
print "\nUsing float2str on complex array
with"::$format::"\n"
float2str(y,$format)

y = complex(x,-x)
$format = "%+8.0f"
print "\nUsing float2str on complex array
with"::$format::"\n"
float2str(y,$format)
```



floor *Floor of Input Array*

Syntax: **floor**(a) or a.**floor**

Description: For each element in 'a' compute the largest integer not greater than that element, i.e. returns the Greatest Integer Value for each element of the array.

Example: The following illustrates the function.

```
[ready]: floor(3.3::4.5)
row 0 =
  3.0   4.0
```

fmod *Floating-Point Modulus*

Syntax: **fmod**(a,b)

Description: Compute the floating point modulus of every element in the input array.

Example: The following illustrates the function.

```
[ready]: a = (4,6,1)      // create row vector
[ready]: fmod(a+.5,3)
row 0 =
  1.50   2.50   0.50
```

free *Free Variable from Memory*

Syntax: **free** a b ... c

Description: Used to erase a list of already defined variables from memory; also have **free all**, **free** plot[#] plot[#] .... plot[#]. The variable list can contain array variables or string variables. To erase all variables from memory use **free all**. As you delete image variables the total memory available will increase. However, the amount of memory increase may not correspond directly to the size of the object just released. This is due to the fact that MSHELL can have two different variable names sharing the same array structure in memory as long as they have an identical content. To erase all of the variables in the current function's scope only, use **free local**.

ftp_cd *Changes Current FTP Server Directory*

Syntax: **ftp_cd**($dirname)

Description: This function works after an ftp_session_init and changes the current working directory into the user specified one; ".." can be used to change to the parent directory. If successful, it returns 0, otherwise it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

Example: see ftp_list example.

ftp_cwd *Get Current FTP Server Directory*

Syntax: **ftp_cwd**()

Description: This function works after an ftp_session_init and can be used to retrieve the current FTP Server working directory as a string variable. If no connection is established with an FTP Server, it returns an empty string.

Example: see ftp_list example.

ftp_del *Delete a File in the FTP Server*

Syntax: **ftp_del**($filename)

Description: This function works after an ftp_session_init and deletes the specified remote file in the FTP server. Multiple file names can be specified by separating them with new lines. If successful, it returns 0, otherwise it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

ftp_get Download a File from the FTP Server

Syntax: **ftp_get**($remotefile, $localfile)

Description: This function works after an ftp_session_init and downloads the specified remote file from the FTP server assigning the specified local file name to the local copy. Multiple file names can be specified by using new lines as file name delimiters. If successful, it returns 0, otherwise it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

Example: see ftp_list example.

ftp_list List FTP Server Files and Subdirectories

Syntax: **ftp_list**()

Description: This function works after an ftp_session_init and can be used to retrieve as a string variable the list of files and subdirectories of the current FTP Server directory. If no connection is established with an FTP Server, it returns an empty string.

Example: The following example shows most of the FTP functions being utilized on a simple test case.

```
// username and password are not required
// for connecting to the FTP server
s = ftp_session_init("ftp.csc.noaa.gov,21,,")
// whenever the returned status is a non-zero value, an error
occurs
if (s) {
          s.text
          return
}

// changing to a different directory
s = ftp_cd("incoming/ameredith/act")
if (s) {
          s.text
          return
}
print "Current directory is "::ftp_cwd()

// listing subdirectories and files of curtrent directory
ftp_list()

// downloading a file from the FTP server
print "Getting file 'gbrevecnts.txt'... "
s = ftp_get("gbrevecnts.txt", "local.txt")
if (s) {
          s.text
          return
}
print "done"

// closing FTP connection
s = ftp_session_close()
if (s) {
          s.text
          return
}
```

ftp_mkdir Create New Directory in the FTP Server

Syntax: **ftp_mkdir**($newdir)

Description: This function works after an ftp_session_init and creates a new specified subdirectory in the FTP Server. If successful, it returns 0, otherwise it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

ftp_put Upload a File to the FTP Server

Syntax: **ftp_put**($localfile, $remotefile)

Description: This function works after an ftp_session_init and uploads the specified local file to the FTP server assigning the specified remote file name to the remote copy. Multiple file names can be specified by using new lines as file name delimiters. If successful, it returns 0, otherwise it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

ftp_rename Rename a File in the FTP Server

Syntax: **ftp_rename**($newfilename)

Description: This function works after an ftp_session_init and renames the specified file name in the FTP server. If successful, it returns 0, otherwise it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

ftp_rmdir Remove Directory from the FTP Server

Syntax: **ftp_rmdir**($dirname)

Description: This function works after an ftp_session_init and removes the specified subdirectory from the FTP Server. If successful, it returns 0, otherwise it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

ftp_session_close Close Current FTP Session

Syntax: **ftp_session_close**()

Description: Closes the current FTP session (if any was established). If successful, it returns 0, otherwise it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

Example: see ftp_list example.

ftp_session_init Open an FTP Session

Syntax: **ftp_session_init**($params)

where,

$params = “host,port,username,password”

Description: Opens an FTP session with the specified host name using the specified port number (if no port number is specified, port 21 will be used). “username” and “password” are optional if FTP server does not require them for logging in. If successful, it returns 0, otherwise it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.
NOTE: THE ftp_... related functions are only supported in the version of MSHELL that runs under ACT’s REACT.

Example: see ftp_list example.

- G -

gauss N-Dimensional Gaussian Density

Syntax: **gauss**(a,invcov,det)

Description: Given a covariance matrix, this function evaluates the Zero Mean Multi-Dimensional Gaussian Density Function. The gaussian density function is evaluated for each column vector in 'a'.

The actual mathematical expression evaluated for each column vector in 'a' is,

$$Zx_c = \frac{1}{\left[2\pi\right]^{N/2} \cdot \left|\Sigma\right|^{1/2}} \cdot \exp\left(\frac{1}{2} x_c^t \Sigma^{-1} x_c\right),$$

where: Σ is the covariance matrix of the Gaussian density function with dimensions N x N (N is the dimension of the columns in 'a'); x_c is a column vector in the input array (the function is evaluated for each column in 'x'); Σ^{-1} is the matrix inverse of the covariance matrix; and $\left|\Sigma\right|$ is the determinant of the covariance matrix.

The output of this function is a row vector of length equal to number of columns present in the input array 'a'.

geclipto Greater or Equal Clip to

Syntax: **geclipto**(a,tval,newval)

Description: Set all the values in the input array greater than or equal to a selected threshold to a new desired value, where, 'a' is the input array, 'tval' is the threshold value, and 'newval' is the desired new value.

Example: The following illustrates the operation.

```
[ready]: x = (0,3,1)
x
row0 =
0.00   1.00   2.00   3.00
[ready]: y = geclipto(x,2,999)
  0.00   1.00 999.00 999.00
```

geindex Greater than or Equal Index

Syntax: **geindex**(a,b)

Description: Similar to **eqindex**, but it returns an index to all elements in 'a' greater or equal than 'b'.

Example: The following illustrates the operation.

```
[ready]: a = geo(-2,8)
[ready]: geindex(a,16)
row 0 =
  4.00-  0.00i   6.00-  0.00i
```

geo Geometric Series

Syntax: **geo**(a,n)

Description: Generates a vector whose elements are the first 'n' terms of the geometric series of the input scalar, 'a'. It is expected that 'a' be a real or complex scalar. Mathematically, **geo**(a,n) generates the row vector,

$$\begin{pmatrix} a^0 & a^1 & \cdots & a^{n-1} \end{pmatrix}$$

Example: The following illustrates the operation.

```
[ready]: geo(-2,8)
row 0 =
  1.00  -2.00   4.00  -8.00  16.00 -32.00  64.00 -128.00
```

Geo2Cart Geographic to Cartographic conversion

Syntax: **out = Geo2Cart(llgrid,pcontrol)**

Description: This function converts from geographic coordinates to cartographic coordinates and viceversa. There is a script function wrapper called geo2cart[] that simplifies the usage to this function. This function is used extensively by MSHELL to generate cartographic products.

The inputs to this MSHELL function are:

llgrid –complex array with latitude (imaginary part) and longitude values in degrees.

pcontrol – used to control the cartographic projection as described in the USGS projection software. Use pcontrol=1 for forward projection (from geographic to cartographic), or -1 for inverse projection. Pcontrol.text contains all the detail controlling parameters for the mapping transformation. It follows the USGS Cartographic Projection Program syntax, see Appendix D: Cartographic Library for syntax details.

The output is:

out - complex array with corresponding carthographic values. Real part has x-component, and imaginary part has y-component.

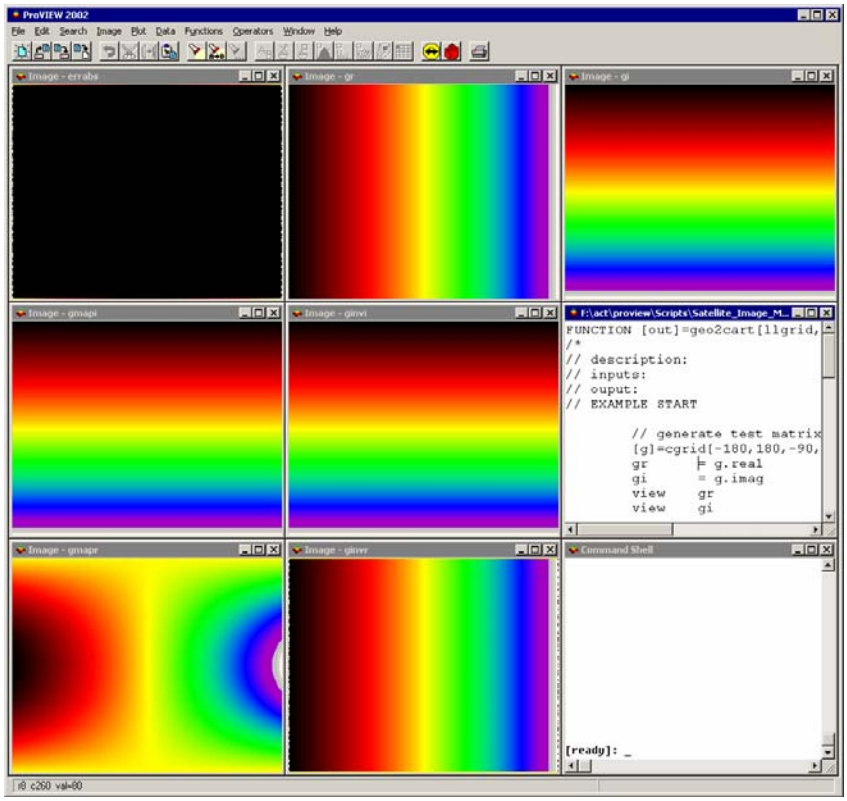
Note: Geo2Cart, makes direct use of a DLL based on the Cartographic Projection Program (developed by U.S. Geological Survey) version 4.4.3,

See *Cartographic Projection Procedures for the UNIX Environment---A User's Manual,* (Evenden, 1990, Open-file report 90-284).
A web reference can be found at: **http://www.remotesensing.org/proj/**

See Also: Geo_cs2cs().
Some of the function using this Geo2Cart and Geo_cs2cs() are: dworld(), and MI_geo2cart.msf, Map_Image.msf, …

Example: This function can also be accessed using the external function script called '**geo2cart.msf'**. This script acts as a trivial wrapper around Geo2Cart.

```
// EXAMPLE START
          // generate test matrix of geographic data points
          [g]=cgrid[-180,180,-90,90,1,1]
          gr  = g.real
          gi  = g.imag
          view    gr
          view    gi
          wtile

          // construct controlling parameters for the mapping
transformation
          $LF     = "\n"
          $control    = "proj=sinu"       ::$LF::     \\
                    "ellps=clrk66"    ::$LF::     \\
                    "lat_0=00"        ::$LF::     \\
                    "lon_0=00"        ::$LF::     \\
                    "no_defs"
          cntl        = 1 // forward transformation
          cntl.text       = $control
          [gmap]      = geo2cart[g,cntl]
          gmapr       = gmap.real
          gmapi       = gmap.imag
          view gmapr
          view gmapi

          cntl        = -1    // inverse transformation
          cntl.text   = $control
          [ginv]      = geo2cart[gmap,cntl]
          ginvr       = ginv.real
          ginvi       = ginv.imag
          view    ginvr
          view    ginvi
          errabs  = abs(g-ginv)

          view    errabs
          wtile
// EXAMPLE END
```



There are a large number of projections supported by Geo2Cart. A list of projections can be obtained from the following command:

```
$cmds = "proj -l "
[result]=syscall[$cmds]
result.text
```

i.e.,

aea	:AlbersEqualArea
aeqd	:AzimuthalEquidistant
airy	:Airy
aitoff	:Aitoff
alsk	:Mod.StererographicsofAlaska
apian	:ApianGlobularI
august	:AugustEpicycloidal
bacon	:BaconGlobular
bipc	:Bipolarconicofwesternhemisphere
boggs	:BoggsEumorphic
bonne	:Bonne(Wernerlat_1=90)
cass	:Cassini
cc	:CentralCylindrical
cea	:EqualAreaCylindrical
chamb	:ChamberlinTrimetric
collg	:Collignon
crast	:CrasterParabolic(PutninsP4)
denoy	:DenoyerSemi-Elliptical
eck1	:EckertI
eck2	:EckertII
eck3	:EckertIII
eck4	:EckertIV
eck5	:EckertV
eck6	:EckertVI
eqc	:EquidistantCylindrical(PlateCaree)
eqdc	:EquidistantConic
euler	:Euler
fahey	:Fahey
fouc	:Foucaut
fouc_s	:FoucautSinusoidal
gall	:Gall(GallStereographic)
gins8	:GinsburgVIII(TsNIIGAiK)
gn_sinu	:GeneralSinusoidalSeries
gnom	:Gnomonic
goode	:GoodeHomolosine
gs48	:Mod.Stererographicsof48U.S.
gs50	:Mod.Stererographicsof50U.S.
hammer	:Hammer&Eckert-Greifendorff
hatano	:HatanoAsymmetricalEqualArea
imw_p	:InternationalMapoftheWorldPolyco
nic	
kav5	:KavraiskyV
kav7	:KavraiskyVII
labrd	:Laborde
laea	:LambertAzimuthalEqualArea
lagrng	:Lagrange
larr	:Larrivee
lask	:Laskowski
latlong	:Lat/long(Geodetic)
longlat	:Lat/long(Geodetic)
lcc	:LambertConformalConic
leac	:LambertEqualAreaConic
lee_os	:LeeOblatedStereographic
loxim	:Loximuthal
lsat	:SpaceobliqueforLANDSAT

mbt_s	:McBryde-ThomasFlat-PolarSine(No.1)
mbt_fps	:McBryde-ThomasFlat-PoleSine(No.2)
mbtfpp	:McBride-ThomasFlat-PolarParabolic
mbtfpq	:McBryde-ThomasFlat-PolarQuartic
mbtfps	:McBryde-ThomasFlat-PolarSinusoidal
merc	:Mercator
mil_os	:MillerOblatedStereographic
mill	:MillerCylindrical
mpoly	:ModifiedPolyconic
moll	:Mollweide
murd1	:MurdochI
murd2	:MurdochII
murd3	:MurdochIII
nell	:Nell
nell_h	:Nell-Hammer
nicol	:NicolosiGlobular
nsper	:Near-sidedperspective
nzmg	:NewZealandMapGrid
ob_tran	:GeneralObliqueTransformation
ocea	:ObliqueCylindricalEqualArea
oea	:OblatedEqualArea
omerc	:ObliqueMercator
ortel	:OrteliusOval
ortho	:Orthographic
pconic	:PerspectiveConic
poly	:Polyconic(American)
putp1	:PutninsP1
putp2	:PutninsP2
putp3	:PutninsP3
putp3p	:PutninsP3'
putp4p	:PutninsP4'
putp5	:PutninsP5
putp5p	:PutninsP5'
putp6	:PutninsP6
putp6p	:PutninsP6'
qua_aut	:QuarticAuthalic
robin	:Robinson
rpoly	:RectangularPolyconic
sinu	:Sinusoidal(Sanson-Flamsteed)
somerc	:Swiss.Obl.Mercator
stere	:Stereographic
tcc	:TransverseCentralCylindrical
tcea	:TransverseCylindricalEqualArea
tissot	:Tissot
tmerc	:TransverseMercator
tpeqd	:TwoPointEquidistant
tpers	:Tiltedperspective
ups	:UniversalPolarStereographic
urm5	:UrmaevV
urmfps	:UrmaevFlat-PolarSinusoidal
utm	:UniversalTransverseMercator(UTM)
vandg	:vanderGrinten(I)
vandg2	:vanderGrintenII
vandg3	:vanderGrintenIII
vandg4	:vanderGrintenIV
vitk1	:VitkovskyI
wag1	:WagnerI(KavraiskyVI)
wag2	:WagnerII
wag3	:WagnerIII
wag4	:WagnerIV
wag5	:WagnerV
wag6	:WagnerVI
wag7	:WagnerVII
weren	:WerenskioldI
wink1	:WinkelI
wink2	:WinkelII
wintri	:WinkelTripel

For additional details on each projection see page 306.

Geo_cs2cs geo coordinate system transformation

Syntax: out = Geo_cs2cs (proj4IN,proj4OUT,xy,direction)

Description: This function converts from the input array 'xy' from one coordinate system to another. The coordinate system must be provided by proper PROJ4 string.

proj4IN - scalar with text attribute set to the projection string that describes the input

proj4OUT - scalar with text attribute set to the projection string that describes the input

direction – if direction=1 ➔ forward transformation, i.e IN to OUT, otherwise it will perform OUT to IN

xy - input coordinate values. Real part has x-component, and imaginary part has y-component. The units of xy most be consistent with the units of the projection space that it represents.

out - output array with transformed values

Note: see also Geo2Cart()

getenv Get Environment Variable

Syntax: **getenv**($string)

Description: Returns a string which is the currently held value for the $string passed in parenthesis. This $string is the environment variable for which information is to be retreived.

```
[ready]: getenv("PATH")
C:\WINNT40\SYSTEM32;
```

getline Finds Line Matching String Pattern

Syntax: **getline**($str,$pattern)

Description: Given a string variable, $str, this function can be used to find the line where the string $pattern appears for the first time.

Example: The following example illustrates the function.

```
[ready]: $a = "Now \n is the \n time \n for all..."
[ready]: $a
Now
 is the
 time
 for all...
[ready]: line = getline($a,"time")
[ready]: $b = $a(line,:)
[ready]: $b
 time
```

See Also: eqindexS(), getpos(), .

getMyIP Get the local IP addresses

Syntax: **getMyIP**()

Description: This function can be used to retrieve the local IP addresses as a new line delimited string variable.
NOTE: This function is only supported in the version of MSHELL that runs under ACT's REACT.

Example: The following example illustrates the function.

```
[ready]: $ip = getMyIP()
[ready]: $ip
192.168.0.1
212.48.10.150
```

getpos Finds String Position Within a Line

Syntax: **getpos**($str,$pattern)

Description: Given a string variable, $str, this function can be used to find the position within a *line where the string $pattern appears for the first time.*

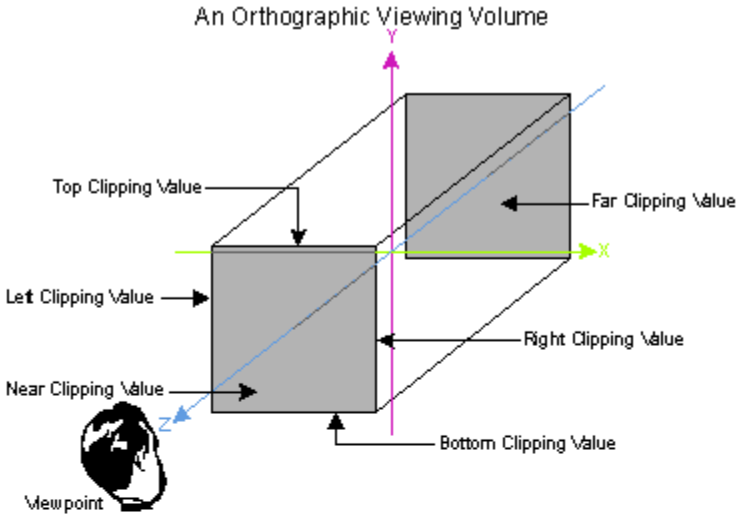
Example: The following example illustrates the function.

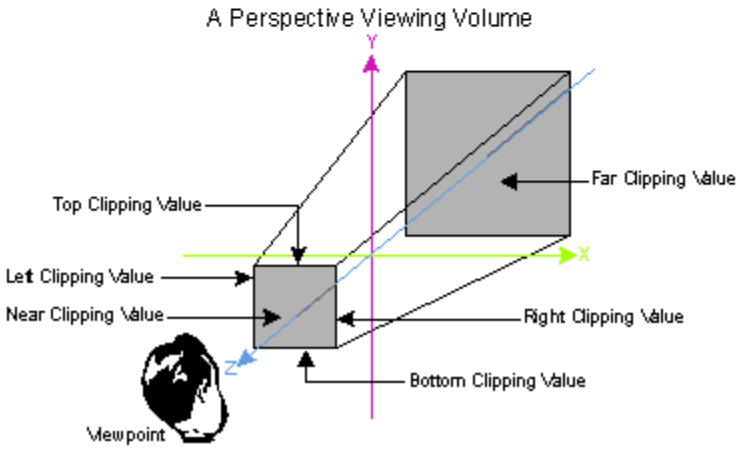
```
[ready]: $a = "Now \n is the \n time \n for all..."
[ready]:$b = $a(2,:)
[ready]:$b
 time
[ready]: getpos($b,"i")
   2
[ready]: getpos($b,"e")
    4
```

glflyover generate perspective view of terrrain

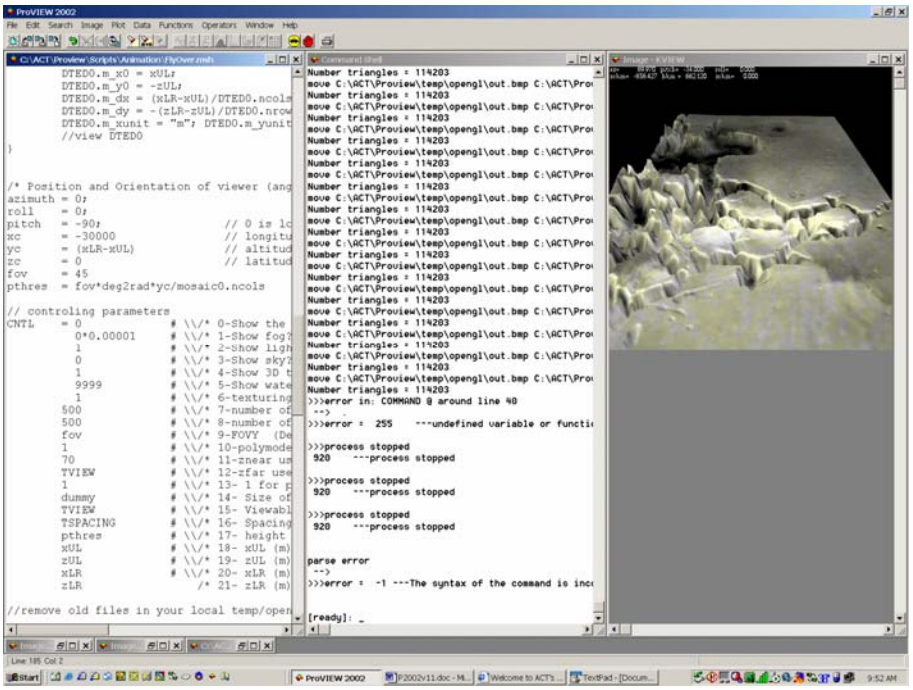
Syntax: **glflyover(DTED,mosaic,CNTL,dummy,PandO)**

Description: Generates a perspective view of an image (mosaic) using input terrain elevation (DTED). This functions can be better understood by looking at the 'flyover.msh' script file. This function is implemented using OpenGL's implementation under the respective OS. The coordinate system used is depicted below





Example:



---

gmtimenow current GMT time

Syntax: **gmtimenow()**

Description: Returns the GreenWich mean time.
Columns 0-5 correspond to year,month,day,hour,minute,second. Column 6 is the day of the week (Sunday=0). Column 7 is the Julian day. If x=gmtimenow(), then x.text will have a nicely formatted time string.

Example:

```
t = gmtimenow()
t

(10^0) X
row 0 =
  2004.00      7.00      8.00     14.00     45.00     53.33
4.00
   190.00      0.00

t.text
2004-07-08T14:45:53.328
```

gtclipto Greater than Clip to

Syntax: **gtclipto**(a,tval,newval)

Description: Set all the values in the input array above a selected threshold to a new desired value, where 'a' is the input array, 'tval' is the threshold value, and 'newval' is the desired new value.

Example: The following illustrates the operation.

```
[ready]: x = (0,3,1)
x
row0 =
0.00   1.00   2.00   3.00
[ready]: y = gtclipto(x,2,999)
  0.00   1.00 2.00  999.00
```

gtindex Greater than Index

Syntax: **gtindex**(a,b)

Description: Similar to **eqindex**, but it returns an index to all elements in 'a' greater than 'b'.

Example: The following illustrates the operation.

```
[ready]: a = geo(-2,8)
[ready]: a
  1.00  -2.00   4.00  -8.00  16.00 -32.00  64.00 -128.00
[ready]: gtindex(a,8)
  4  + 0i   6 + 0i
```

- H-

hammiw Hamming Window

Syntax: **hammiw**(n [,m])

Description: Generates a 1-D or 2-D Hamming Window, where the *n*th element in the Hamming Window, is defined as,

$$w_n = 0.54 - 0.46\cos(\frac{2\pi}{N} \cdot n), \quad n = 0,1,2,\ldots N-1$$

This window has side lobes in the frequency domain of -43dB. Note, the Hamming Window is a special case of the Blackman-Harris Window. The syntax for generation of a row vector containing the 1-D, n element Hamming Window is, **hammiw**(n), and the syntax for generating the 2-D n x m element Hamming Window is, **hammiw**(n,m). This 2-D array is equivalent to the MSHELL construction:

```
hammiw(n)' * hammiw(m)
```

Example: Let 'a' be an array with dimensions that are a power of two. The following MSHELL command will multiply the array 'a' with a 2-D Hamming Window, followed by the 2-D FFT,

```
fft2(hammiw(nrows(a), ncols(a)) *. a)
```

help Invokes the Help Utility

Syntax: **help** [topic]

Description: You can invoke the help utility directly from the command line. An optional topic argument can be provided to search for help on that topic.

Example: For help on the **cos** function, type from the command line,

```
help cos
```

heqlut Histogram Equalization LUT

Syntax: **heqlut**(a) or a.**heqlut**

Description: Computes the 256 entry (8 bit) look-up-table (or intensity transformation) which when applied to the input image, 'a', will result in a more uniform distribution of intensity value.

Example: An image x can be subjected to the heqlut intensity transformation, prior to display, using the following MSHELL instructions,

```
        wcolut3 = ones(1,3)*heqlut(X)
select wcolut3;
```

Note that selecting wcolut3 does not change the content of the actual image in memory, only the way that it is displayed.

hist Histogram Generator

Syntax: **hist**(a,amin,amax,n)

Description: This is a general purpose histogram generation subroutine. It performs a histogram value of the element in the source input array. The elements on the input array are first transformed by a linear equation which determines the range of the data to histogram, and the number of bins on that range, where 'a' is the input array, 'amin' and 'amax' are the extreme values, and 'n' is the number of levels over which the input will be grouped. This function is particularly useful with floating point data, while "**hist255**", below, is optimized for integer data in the range of 0 to 255.

hist255 Histogram of 8 bit data

Syntax: **hist255**(a) or a.**hist255**

Description: Generate a histogram of the distribution of intensity values in the input array. The data values within the input array must assume only integer values in the range of 0 to 255 inclusive. The returned vector has 256 entries; where a k positive value in the i^{th} -1 entry implies there were k elements in the input array which assumed the i^{th} value.

Example: The following is a typical application.

```
[ready]:x = reada("eqohare.chr","char");
[ready]: plot(hist255(x))
```

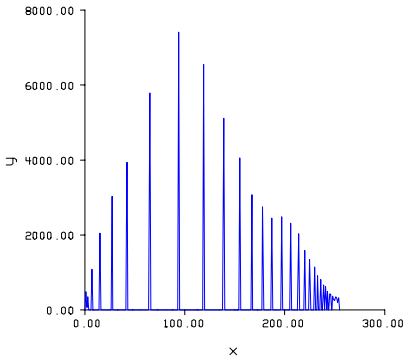


Figure 3

---

hyplut Hyperbolic Histogram LUT

Syntax: **hyplut**(a,imin,imax)

Description: Computes the 256 entries (8bit) look-up-table (or intensity transformation) which when applied to the input image will result in a hyperbolic distribution of intensity values. The input arguments are: the input image, 'a' and the intensity limits 'imin' and 'imax' , respectively, the minimum and the maximum values that any intensity value in 'a' can be map to. Both 'imin' and 'imax' must be between 0 and 255 inclusive. It is expected that the input image is representative of an image with 8 bits/pixel.

An advantage of using a hyperbolic lut is that it accounts for the assumed logarithmic or cube root response of the photoreceptors of the human eye model, resulting in a perceived more uniform distribution of intensity values. The actual intensity mapping utilized has the following mathematical representation:

$$hyplut(k) = i_{\min} \cdot \left(\frac{k_{\max}}{k_{\min}} \right)^{\sum_{j=0}^{K} P_a(j)}$$

$P_a(j)$ = hist255(a)/(a.nrows*a.ncols)

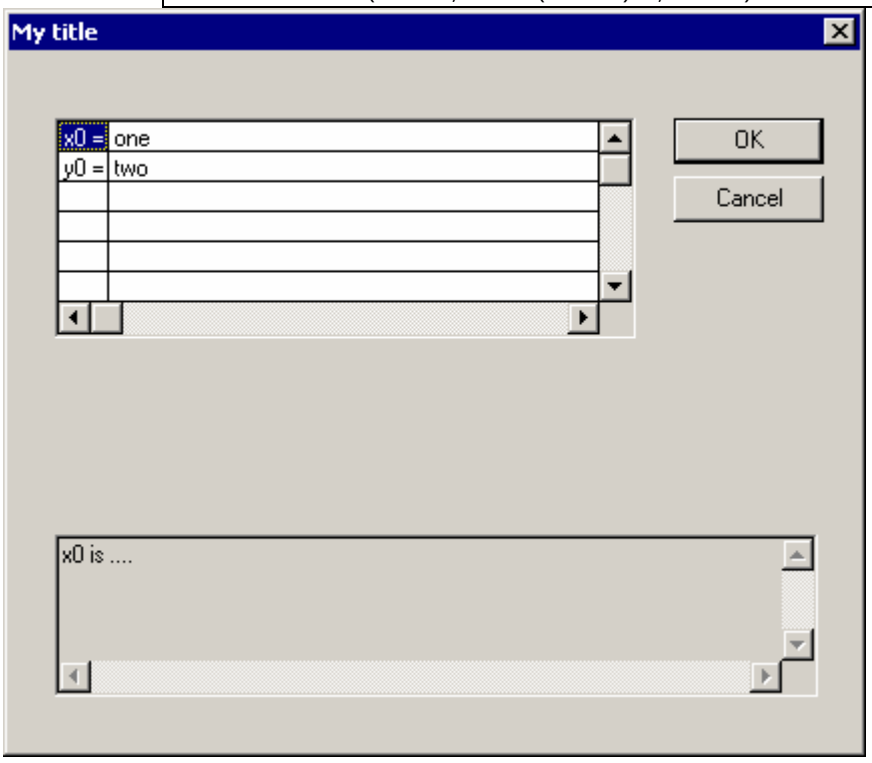
iboxlist Input Box List

Syntax: **iboxlist**($title,nitems,$ilist)

Description: This function is used to create an input box on the screen based on an input list, $ilist, of options. It allows one to modify various options in a form that is display on the screen.

Example: the following illustrates how this function can be used

```
$title =  "My title"
          $ilist = "" //initialize input list

          $item = "x0 ="
          $itemdesc = "x0 is .... "
          $itemval  = "one"
          [list] = add2list[$ilist,$item,$itemdesc,$itemval]
          $ilist = list.text

          $item = "y0 ="
          $itemdesc = "y0 is related to ...."
          $itemval  = "two"
          [list] = add2list[$ilist,$item,$itemdesc,$itemval]
          $ilist = list.text

$list = iboxlist($title,nlines($ilist)+5,$ilist)$list =
```



idct8x8 Inverse Discrete Cosine Transform

Syntax: **idct8x8**(b)

Description: Used to perform the inverse discrete cosine transform on an array "b" which is the DCT of a previous function. The resultant will be identical to the original array previous to the creation of "b", except for small errors resulting from computer round off errors.

ident *Generate an Identity Array*

Syntax: **ident**(n)

Description: Create an N x N array in memory in which the main diagonal elements are set to value **+1** and all off diagonal elements are set to zero.

Example: The following MSHELL statement will create a 4x4 identity array in memory,

```
[ready]: ident(4)
row 0 =
   1.00    0.00    0.00    0.00
row 1 =
   0.00    1.00    0.00    0.00
row 2 =
   0.00    0.00    1.00    0.00
row 3 =
   0.00    0.00    0.00    1.00
```

IF-ELSE *IF-ELSE Conditionals*

Syntax: ***if****(expression){*

statements

}

or

if*(expression){*

statements

}else{

statements

}

Description: This condition is used to execute the statements if the expression is considered true.

ifft Inverse 1-D FFT

Syntax: **ifft**(a) or a.**ifft**

Description: Compute the one dimensional inverse Fourier transform of the input array. It is expected that the row dimension of the input array is a power of two. Note that the output of this operation is a complex array.

ifft2 Inverse 2-D FFT

Syntax: **ifft2**(a) or a.**ifft2**

Description: Compute the two dimensional inverse Fourier transform of the input array. It is expected that the dimensions of the input array are powers of two. Note that the output of this operation is a complex array.

imag Imaginary Part

Syntax: **imag**(a) or a.**imag**

Description: For a complex input array, extract the imaginary part for each element in the array.

Example: The following MSHELL statement will extract the imaginary part of the 2D Fourier Transform of 'a', and store the result in 'c',

```
[ready]: c = imag( fft2(a) )
```

image2surface project planar image onto a surface

Syntax: **image2surface**(image,xy,z,cntl1,cntl2,cntl3)

Description: Given an input 'image' defined over a rectangular grid, this command will drape the input image intensities on a surface grid defined by xy&z.
Note: This function is **primarily used** in MSHELL to go from sensor projection or sersor-scan into a cartographic map representation when a dense rectangular grid of ground control points is available, (in this case z is set to a constant value of zero and cntl3=0).

image	Real array with amplitude values for the image
xy	xy = complex (x,y) with values to be used for the projection. The number of rows and columns in xy must match the dimensions of image. If set to the zero scalar, a rectangular grid of xy, with monotonically increasing values, will be internally generated.
z	Real array with height values to be used in the projection. The number of rows and columns in xy **must** match the dimensions of image.
cntl1	**cntl1=ns::outrows::outcols::non_valid_pixel::background_level** Where, **ns**=1,…8 used to control the sampling of the input xyz grid. ns=1 is most accurate and ns=8 is the least accurate. **outrows** – is the number of rows in the desired output image **outcols** – is the number of columns in the desired output image **non_valid_pixel** – amplitude value used to specify a pixel with no data in 'image'. Such pixels will not be used in the projection **background_level** – initialize output image to this value, before starting the projection process
cntl2	Cntl2 = 0 ➔ compute range of min and max values internally Cntl2 = xmin::xmax::ymin::ymax::zmin::zmax force the use of the provided range of min and max values.
cntl3	Cntl3 = 0 ➔ to map input image into a planar grid with not additional rotation or tilt transormations or cntl3 = rot::tilt::show_base_image::show_borders for orthographic projections where, rot – is rotation of the input image in degrees tilt – is a rotation of the input image away from the normal show_base_image and show_borders must be set to zero. If this option of cntl3 is used, then cntl2 is set to 0 automatically.

See also:
Map_Image[ ], Map_Images[ ], MI_geo2cart[], … - these are external functions that supports intensity mapping from geographic to cartographic.

Example: The following example illustrates the function.

```
// Example1: illustrate image2surface with different sampling parameters
truth  = reada("eqohare.chr","char")// generate test sensor projection image
[truth]=zoom2fit[truth,300,300]
image0 = truth
view truth
[xy0]  =cgrid[0,image0.ncols-1,0,image0.nrows-1,1,1]
x0     = xy0.real
y0     = xy0.imag
k      =5
[xsp]=spiral[x0,k/10*3]
[ysp]=spiral[y0,k/10*3]
[isp]=spiral[image0,k/10*3]
view   xsp; view ysp; view   isp
wtile

// now apply image2surface, while changing  xy-sampling
image0 = isp

j      = sqrt(-1)
xy     = xsp+j*ysp

z      = zeros(image0.nrows,image0.ncols)*0
non_valid_pixel=0
groi   = eqindex(isp,non_valid_pixel)
backg  = 0
i      =75
while(i>1){
       xlev   = (0,x0.max-1,i)
       ylev   = (0,y0.max-1,i)
       backg  = 0
       [grid] = DispContours[xy0,xlev,ylev,backg]
       dummy  = image0

       groi2  = index(grid)
       dummy(groi2)=128
       dummy(groi) = non_valid_pixel
       image  = dummy

       view image
       ns     =i
       print "ns = "::int2str(ns)::"\n"
       outrows =grid.nrows
       outcols =grid.ncols

       cntl1  =ns::outrows::outcols::non_valid_pixel::backg
       cntl3  =0
       cntl2  =0
       $ts = M_time
       image(wdef(nint(xsp.nrows/2/i)*i,nint(xsp.ncols/2/i)*i,i,i))=255
       outg   = image2surface(image,xy,z,cntl1,cntl2,cntl3)
       out0   = image2surface(image0,xy,z,cntl1,cntl2,cntl3)
       $te = M_time
       $ts::$te
       view  outg;    view  out0
       wtile
       i = i-3
}
```

```
// EXAMPLE 2:  reading read lunar topography from Clementine
data
luntopo = reada("luntopo.flt","float")
B = luntopo.scale255  // set texture image to topography
view B

luntopo    = scale255(luntopo)
view luntopo

cntl1     =ns::outrows::outcols::non_valid_pixel::backg
cntl2     =0
cntl3     =30::40::0::1

moonTopo = image2surface(B,0,luntopo/2,cntl1,cntl2,cntl3)
view255 moonTopo
```

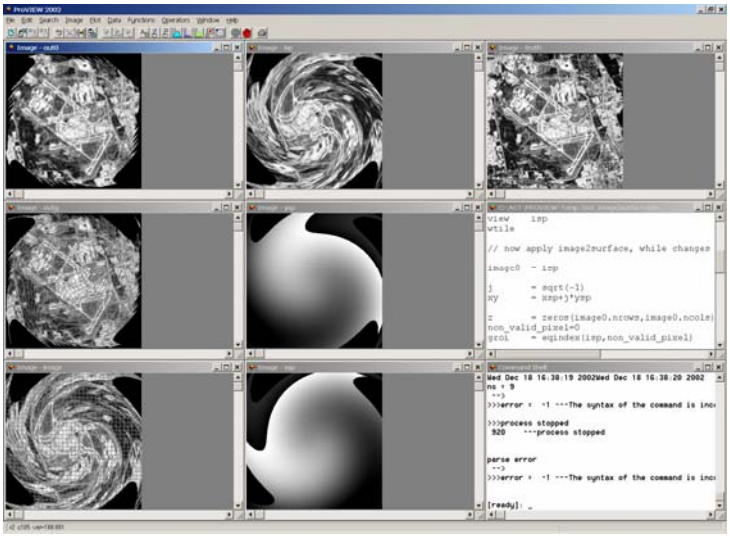


Figure 5 Example #1 output

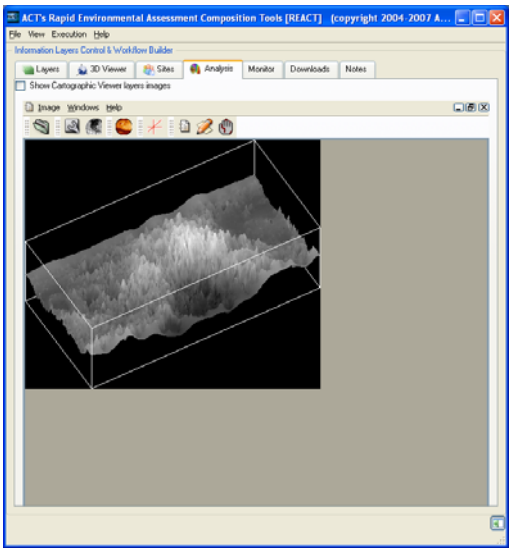


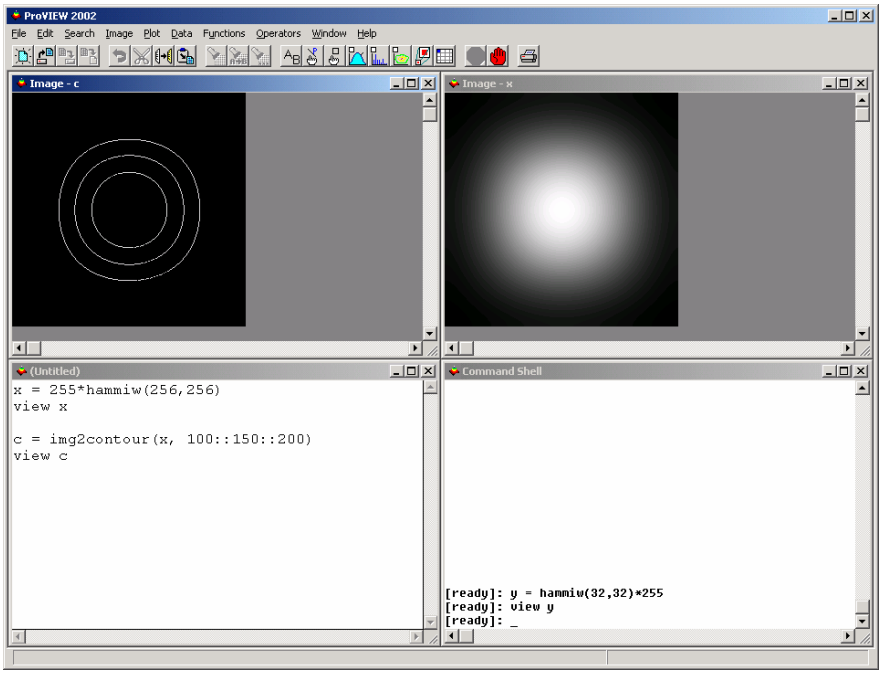
Figure 6 Example #2 output

img2contour Create a contour image

Syntax: **img2contour**(image, levels # graylevel)

Description: Generates an output contour from the input image. The levels are a row vector describing the contour steps desired, and the graylevel is a row vector describing the value to use when drawing the contour line.

See Also:



include Invoke an MSHELL Script File

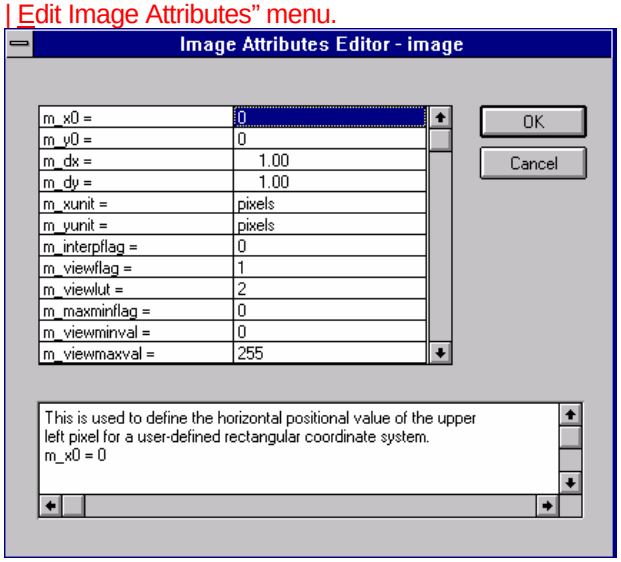
Syntax: **include** "fname"

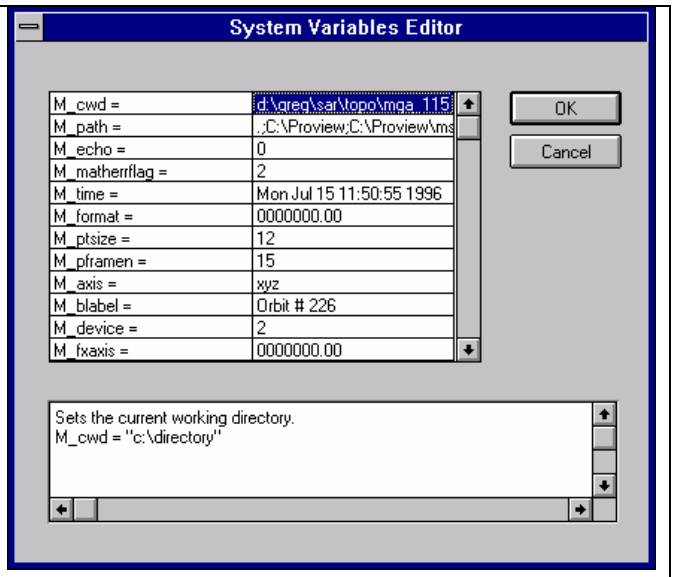
Description: A file with a sequence of MSHELL commands can be invoked at any time through the use of the include command, where “fname” is the name of the script file to be executed.

Example: Given an MSHELL script file named ‘test.msh’, the MSHELL instructions contained in the file can be executed by simply typing: **include** "test" or **include** "test.msh".

In the first case above, the extension ".msh" was not included in the argument. In that case MSHELL will try to open the file 'test', if that fails it will try again after adding the extension '.msh'. The system variable M_path establishes which directories, in addition to the present directory, are to be searched.

The following are some useful script files which were not coded as script functions:

Script File	Description
file2pds.msh	This script file is used to convert an existing image file to a PDS image file whereby the new image file now contains a PDS keyed header.
imgedit.msh	This script file is used for the editing of image files. It is the same script which is called when one uses the "Image | Edit Image Attributes" menu. 
img2pds.msh	This script file is used to convert open images to PDS images whereby the new image now contains a keyed header.
mpeg.msh (outdated)	This script file is used for the creation of MPEG movies from a group of existing similar image files. This script prompts the user for any needed information and gives the capability of advanced functions or beginner use.
pdsmap.msh	This script file is used to map several or just one PDS formatted image onto a lat-long annotated grid.
sysedit.msh	Allows to edit basic system variables from a drop down box.


System Variables Editor
M_cwd =
d:\greg\sar\topo\mga_115
M_path =
.;C:\Proview;C:\Proview\ms
M_echo =
0
M_matherrflag =
2
M_time =
Mon Jul 15 11:50:55 1996
M_format =
0000000.00
M_ptsize =
12
M_pframen =
15
M_axis =
xyz
M_blabel =
Orbit # 226
M_device =
2
M_fxaxis =
0000000.00
OK
Cancel
Sets the current working directory.
M_cwd = "c:\directory"

index *Index of Non-Zero Elements*

Syntax: **index**(a,b)

Description: Similar to **eqindex**, but it returns an index to all elements different than zero.

Example: The following illustrates the indexing.

```
[ready]: a = (-2,2,2,2)
row 0 =
 -2.00   0.00   2.00
row 1 =
 -2.00   0.00   2.00
[ready]: index(a)
row 0 =
  0.00 -0.00i    2.00 -0.00i    0.00 +1.00i    2.00 +1.00i
```

input *waits user input*

Syntax: **input** a

Description: Pauses for input from console (does not work in GUI, only console MSHELL), then assigns the input value to string or array variable.

inputbox *Prompt User for Input*

Syntax: **inputbox**($prompt, $title, $default)

Description: Prompts for input through a dialog box, where all the input arguments are strings. This function returns a string which can be convert to a number. **See Also:** **str2int** and **str2float**

Example: The following illustrates the instructions and the Dialog Box.

```
[ready]: inputbox( "enter new value", "INPUT MENU", "1")
```

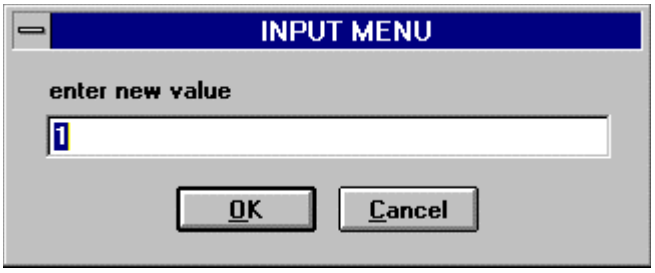


Figure 4

---

inputfocus *Array Variable with Current Focus*

Syntax: x = **inputfocus**

Description: Sets the defined variable equal to the variable whose window currently has focus.

See Also: M_inputfocus

---

int *Integer part*

Syntax: **int**(a) or a.**int**

Description: Computes the integer part for each element in the input array, 'a'.

---

int2str *Convert an Integer to a String*

Syntax: **int2str**(a)

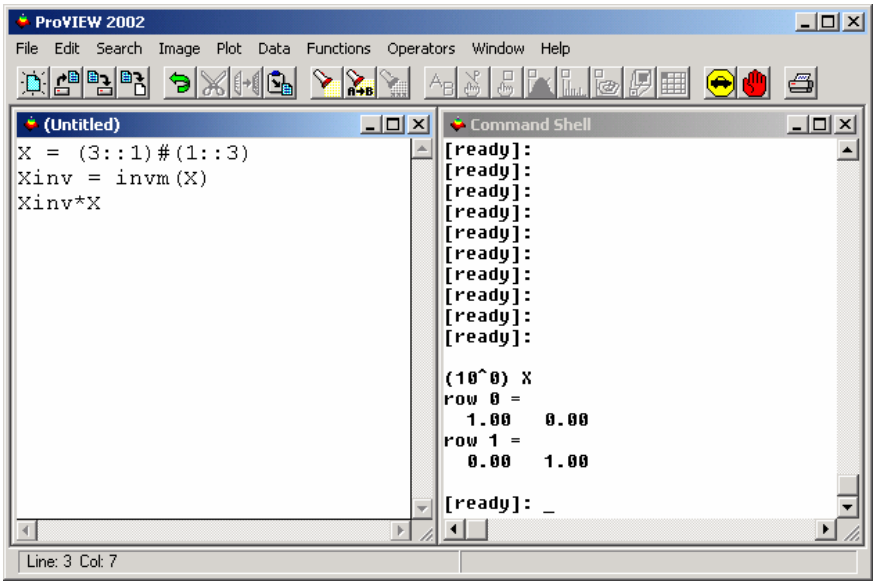
Description: Converts the input scalar to an integer number and then to a string.

Example:

invm *Inverse of an Array*

Syntax: **invm**(a)

Description: Computes the inverse of matrix "a", so that "a" multiplyed by its inverse equals the identity matrix.



itoa *Integer to Ascii (same as int2str)*

Syntax: **itoa(**x**)**

Description: Converts the integer ‘x’ to a string. See ‘int2str’.

- L -

ladd2groi local add of constant to selected pixels

Syntax: **ladd2groi(**X,groi,V**)**

Description: Increments every pixel in X(groi) by scalar (or row vector) 'V'. **This function permanently modifies the content of X**, The following MSHELL instruction make use of this function with internal checking for not corrupting shared arrays.

a(groi)++

a+=b

a(groi)+=b

Hence try to use the above as opposed to directly using ladd2groi.

Example: The following illustrates the operation.

```
x         = (0,10,1)
x2        = x^2
groi      = rindex(x,5,8)
b         = -x(groi)^2

print "x ="::float2str(x,"%7.0f")::"\n"

a1        =   x
a1        +=  x2
print "a1="::float2str(a1,"%7.0f")::"\n"
a2        =   x
a2(groi)+=    1
print "a2="::float2str(a2,"%7.0f")::"\n"
a3        =   x2
a3(groi)+=    b
print "a3="::float2str(a3,"%7.0f")::"\n"
a4        = x2
a4(groi)+=    -x(groi)^2
print "a4="::float2str(a4,"%7.0f")::"\n"
```

The above code generates the following output:

x = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

a1= 0, 2, 6, 12, 20, 30, 42, 56, 72, 90, 110

a2= 0, 1, 2, 3, 4, 6, 7, 8, 9, 9, 10

a3= 0, 1, 4, 9, 16, -600, -1260, -2352, -4032, 81, 100

a4= 0, 1, 4, 9, 16, 0, 0, 0, 0, 81, 100

leclipto Lower or Equal Clip to

Syntax: **leclipto**(a,tval,newval)

Description: Set all the values in the input array, 'a', that are less than or equal to a selected threshold value, 'tval', to a new desired value, 'newval'.

Example: The following illustrates the operation.

```
[ready]: a = randu(1,7)
[ready]: a
row 0 =
  0.43   0.43   0.57   0.09   0.78   0.56   0.74
[ready]: leclipto(a,.50,1)
row 0 =
  1.00   1.00   0.57   1.00   0.78   0.56   0.74
```

leindex Lower or Equal Index

Syntax: **leindex**(a,b)

Description: Similar to **eqindex** , but returns an index to all elements in 'a' lower than or equal to 'b'.

Example: Using the array 'a', previously defined in **leclipto**,

```
[ready]: a = randu(1,7)
[ready]: leindex(a,.5)
row 0 =
  0.00  -0.00i   1.00  -0.00i   3.00  -0.00i
```

linterp Linear Interpolation

Syntax: **linterp**(f,x)

Description: Linearly interpolates between data points located on a one dimensional grid. The function argument, 'f', is a 1-dimensional row vector with an assumed interpixel distance of 1 along the axis, i.e. the element values of 'f' can be considered the ordinate values for some function over a set of increasing (or decreasing) abscissa values. The argument 'x' is a real 1-Dimensional row vector

containing values of the abscissa for which ordinate values are to be linearly interpolated. For every point that extrapolation is attempted the result is set to zero.

Example: The following two examples illustrate the function.

```
[ready]: f = hammiw(1,4)
[ready]: f
(10^0) X
row 0 =
  0.01   0.04   0.08   0.04   // assumed abscissa values of:
0, 1, 2, 3
[ready]: x = (-1,7,1) * 0.5
[ready]:  x
 -0.50   0.00   0.50   1.00   1.50   2.00   2.50   3.00
3.50
[ready]: linterp(f,x)
  0.00   0.01   0.02   0.04   0.06   0.08   0.06   0.04
0.00
[ready]: y = 0.2
[ready]: linterp(f,y)
  0.01376
```

load Loads saved arrays or strings from a file

Syntax: **load** $savefile

Description: Loads arrays or strings from the $savefile.

See Also: **save**

loadDLL Loads a DLL for Execution

Syntax: **loadDLL**($pathname)

Description: Used to load a DLL for later execution using callDLL. The path and name of the DLL must be passed as a string.

Example: The following will load the “delaunay.dll”. Notice that the path to this file has also been included.

```
[ready]: loadDLL("c:/proview/bin/delaunay.dll")
```

LOCAL Declare a locally scoped variable

Syntax: **LOCAL** my_array

Description: Declares a variable local to the MSF function. Only necessary when in the **LOCALOFF** mode.

LOCALON Turn on locally scoped variables

Syntax: **LOCALON**

Description: Used in an MSF function. Variables referenced after this command will **NOT** affect top level variables, but will reference local function variables. By default MSF functions are in the LOCALON state.

LOCALOFF Turn off locally scoped variables

Syntax: **LOCALOFF**

Description: Used in an MSF function. Variables referenced after this command will affect variables defined outside of the local function, instead of referencing local function variables. Use with caution!! By default MSF functions are in the LOCALON state.

localtimenow Local Time Now

Syntax: **localtimenow()**

Description: Returns the current system time as a column vector. Columns 0-5 correspond to year,month,day,hour,minute,second. Column 6 is the day of the week (Sunday=0). Column 7 is the Julian day. If x=localtimenow(), then x.text will have a nicely formatted time string.

Example:

```
ready: t = localtimenow()
ready: t

(10^0) X
row 0 =
  2004.00      7.00      8.00     10.00     43.00     59.58
4.00
   190.00      1.00

ready: t.text
2004-07-08T10:43:59.578
```

see also: gmtimenow()

---

log Natural Logarithm

Syntax: **log**(a) or a.**log**

Description: Computes the natural logarithm of each array element. The actual mathematical expression computed is given by,

$$c_{j,i} = \log(a_{j,i}) \quad \text{for all j, i;}$$

where j and i are row and column indices respectively.

The **Log** of zero is not defined and will generate an error, where as the **Log** of a negative number will generate a complex number.

Example: The following MSHELL statement will compute the natural logarithm of 'a' and store the result in 'c',

```
[ready]: c = log(a)
```

---

log10 Base 10 Logarithm

Syntax: **log10**(a) or a.**log10**

Description: Compute the base 10 logarithm of each array element. The **Log10** of zero is not defined and will generate an error, whereas the **Log10** of a negative number will generate a complex number. The actual mathematical expression computed is given by,

$$c_{j,i} = \log_{10}(a_{j,i}) \quad \text{for all j, i;}$$

where j and i are row and column indices respectively.

Example: The following MSHELL statement will compute the base 10 logarithm of 'a' and store the result in 'c',

```
[ready]: c = log10(a)
```

ltclipto Lower than Clip to

Syntax: **ltclipto**(a,tval,newval)

Description: Set all the values in the input array, ‘a’, below a selected threshold value, ‘tval’, to a new desired value, ‘newval’.

Example: Using the array ‘a’, previously defined in **leclipto**,

```
 [ready]: ltclipto(a,0.5,.0.2)
row 0 =
  0.20   0.20   0.57   0.20   0.78   0.56   0.74
```

lthresh Less than Threshold

Syntax: **lthresh**(a,b)

Description: This sets all values within "a" which are less than the treshold "b" equal to the threshold.

ltindex Lower Index

Syntax: **ltindex**(a,b)

Description: Similar to **eqindex** , but it returns an index to all elements in ‘a’ lower than ‘b’.

Example: Using the array ‘a’, previously defined in **leclipto**,

```
[ready]: ltindex(a,.5)
row 0 =
  0.00-  0.00i   1.00-  0.00i   3.00-  0.00i
```

Look-up-Table LUT associated to image

Syntax: **x.LUT** and **x.LUTdef**

Description: *Note: This capability was incorporated in MSHELL version 3.* Given an image 'x', you can set or read an associated look-up-table to this variable using the 'x.LUT'. Once assigned, if the user sents the image for display (using view or view255) the view command will make use of the assigned LUT.

'x.LUTdef' is used to save a copy of the default palette that gets loaded when the image is read from disk. If the user modifies 'x.LUT' the original state can be restored with 'x.LUT = x.LUTdef'

Example:

```
x = reada("eqohare.chr","char");  // open and image
view x;                           // gray scale palette
y = x;                            // assign x to y
y.LUT = (255,0,1,3)               // assign an inverse
palette to y
view y;                           // inverted palette
z = x;
z.LUT  =
reada(M_proviewdir::"/palettes/apl_wind.asc","ascii"); //
color palette
view z;                           // color palette
```

All the palettes available are located in the 'palettes' directory.

M_ System Variables

Description: System variables are used to control different functions within the MSHELL environment. Some of the MSHELL system variables are strings while others are numbers. All system variables are prefixed by '**M_'**. For example, to initialize the x-axis label to the string "Time" use, **M_xlabel** = "Time". If you create your own M_ variable, e.g. M_mine=3, it will behave as system variable, i.e. it will be accessible from inside any script of function.

Note that system string variables do not required the '$' symbol normally associated with user defined string variables. System variables can be read into other user defined variables, e.g. $message = **M_xlabel**, is a legal construction.

Approximately three quarters of the system variables are used in controlling the **plot** and **plot3d** function output.

The following list categorize and describes the various the system variables.

See **include system files** in this section for **sysedit.msh** which allows the user to change all system variables at one time from a table. Also, under the Edit menu one can find the Edit System Variables item which calls sysedit.msh.

Plot Related Variables

M_axis When using **plot3d** you can control if a numeric axis is drawn or not using the **M_axis** string; (string variable). If **M_axis** is set to "xyz" all the axis will be drawn. If x,y, or z is omitted from the string the corresponding axis is also omitted.

M_blabel Bottom label used in plots; (string variable).

M_device Output plot device, values: 0 = DT2861/AL860HRG, 1 = plot file format, 2 = Windows Screen (default)

M_fxaxis Used to control the number of digits used in the x-axis, see **M_format** for more information.

M_fyaxis Similar to **M_fxaxis** but for the y-axis.

M_fzaxis Similar to **M_fxaxis** but for the z-axis.

M_hidden If set to 0 hidden line removal will not be used in **plot3d**; default value is 1 (use hidden line removal).

M_holdx Sets axis display: if 0 display is the extent of the data, if 1 display is limited by **M_xmin** and **M_xmax**

M_holdy Similar to **M_holdx** but for y.

M_holdz Similar to **M_holdx** but for z.

M_linetype Sets plot line type style. Valid line styles are: 0, for symbol; 1, for solid line; and 2, for spikes.

M_panel If set to 1 a side panel will be used in the **plot3d**; default value is 0.

M_phi1 Select rotation around z axis in **3dplot**: values range between -**90** and **90** (degrees).

M_phi2 Select out of plane angle in **3dplot**: values range between **-90** and **90** (degrees).

M_tlabel Title label; (string variable).

M_xdir If set to 0 only the lines of constant x values will appear in a **plot3d**; default value is 1.

M_xlabel X axis label; (string variable).

M_xlog Log scale of x-axis, valid values are 0, 1, 2, or 3. If set to: 0 the option is disabled; 1, 2, or 3 yield logarithmic scaling with increasing resolution; applicable to **plot** only.

M_xmax Set maximum x-axis value, use **M_holdx** = 0 to cancel, **M_holdx** = 1 to restore.

M_xmin Set minimum x-axis value, use **M_holdx** = 0 to cancel, **M_holdx** = 1 to restore.

M_xnice If set to 1 the max. and min. for the x-axis are computedsuch that a nice set of numbers is selected.

M_xtic Used to control the spacing of tic marks along the x-axis; default value is 1. Only applies if **M_xnice** is set to zero.

M_ydir Similar to **M_xdir** but for y.

M_ylabel Similar to **M_xlabel** but for y.

M_ylog Similar to **M_xlog** but for y-axis.

M_ymax Set maximum y axis value, use **M_holdy** = 0 to cancel, **M_holdy** = 1 to restore.

M_ymin Set minimum y axis value, use **M_holdy** = 0 to cancel, **M_holdy** = 1 to restore.

M_ynice Similar to **M_xnice** but for y-axis.

M_ytic Similar to **M_xtic** but for y-axis.

M_zlabel Similar to **M_xlable** but for z.

M_zmax Maximum z axis value.

M_zmin Minimum z axis value.

M_znice Similar to **M_xnice** but for z-axis.

M_ztic Similar to **M_xtic** but for z-axis.

M_xorigin x offset for the left of the plot output

M_yorigin y offset for the top of the plot output

M_xlen length of x-axis (?? units)

M_ylen length of y-axis (?? units)

M_zlen length of z-axis (?? units)

M_sym point symbols used when M_linetype is zero.

1 = squares with a top center tickmark
2 = circles with a a top center tickmark
3 = triangles with a top center tickmark
4 = Y shaped marks
5 = X shaped marks
6 = diamonds with a top center tickmark
7 = little umbrella looking symbols
8 = little hourglass looking symbols
9 = Z shaped marks
10 = squares, corner ticks and upper right to center tick
11 = asterisks
12 = plus symbols
13 = diamonds

M_height Font height of plot axis annotations

M_panel For plot3d, draw vertical lines on the front panels

M_plotcolor graylevel (0-255) of plotting output

M_ndigits **reserved**

M_xdigits **reserved**

M_ydigits **reserved**

M_zdigits **reserved**

M_scalef **reserved**

M_deltac **reserved**

M_deltar **reserved**

M_digits **reserved**

M_contour_view **reserved**

M_edit **reserved**

Text Output

M_format Controls the number of digits used when printing values to the Command Line Window screen. For example, **M_format** = "00.000" specifies that the largest value that can be represented, excluding the exponent, is 99.999. Anything larger than this is printed as ##.###.

M_ifont used to specify image font type (text2image)Currently 'times' is the only option

M_ptsize Controls the font size to be used when overlaying text to images using the text2image commands; the available point sizes are: 9, 10, 12, 14, 18, and 24.

Time Related Variables

M_time Returns the system time; this variable is read only, no value is needed or assigned.

```
[ready]: M_time
Tue Apr 25 12:04:35 1995
```

System Related Variables

M_cwd Sets the current working directory. For example, to set 'c:/mydir' as the current working directory type:
M_cwd = "c:/mydir"

M_path The path string assigned to this variable will be searched when trying to open an input file. For example, to set **M_path** to the directorys 'c:/mshell/bin' & 'c:/mshell/msf' type: **M_path** = "c:/mshell/bin; c:/mshell/msf".

M_wdb Contains word-data-base file to use when drawing coast-lines using the dworld() function.

M_windir This is the windows root directory.

M_echo This toggles the display of executed commands within a script file.
M_echo=10 causes all commands to be displayed as they are performed.
M_echo=5 is nominal output.
M_echo=0 will prohibit all printing, including explicit print statements.

M_prompt Sets the command prompt text (MSHELL console mode only)

M_windir Shows the WINDOWS Root directory (WIN32 Only)

M_proviewdir Shows the PROVIEW root directory

M_errorscript contains name of error script to execute in the event of an error. If empty, no error script is executed. By default, MSHELL starts 'M_errorscript' empty.

M_errorstring String output of last error encountered

M_version The ProVIEW/MSHELL version number.
Note: After version 3, significant changes in the handling of the color palette where introduced via the 'x.LUT' image attribute.

M_nprint Entering this alone will send a screenshot to the current default printer (GUI only)

M_inputfocus provides name of image that has the present or last input focus of the GUI

M_xpix Obsolete, see array window operators m_viewx0 and m_viewy0.

M_ypix Obsolete, see array window operators m_viewx0 and m_viewy0.

M_logmode 0=send output to STDOUT only (default)
1=send output to M_logfile
2=send output to STDOUT and M_logfile

M_logfile used as the output file (including path)

M_OS provides the operating system under which the interpreter is running (UNIX or Windows_NT)

M_processid provides process id on current session

M_execution_mode
provides execution mode for the version of the executable that is using MSHELL, e.g. GUI, or CGI, or CONSOLE.

CGI mode

M_cgi 1 means running in cgi model, 0 otherwise.

M_ipaddr Only in web-based mode, show the IP addr of the client

M_wwwroot This is the root www directory, which is used when in cgi mode.

M_xtype Contains the requested output document type HTTP header (Content-type). This must be set when provweb.exe is called (i.e. http://server.com/cgi-bin/provweb.exe?_xtype=text/html ...) Note the variable name there is _xtype. The output behaviour is as follows:

Blank – ProVIEW-Web begins a "text/html" document, with fixed font formatting (<HTML><PRE>…) If you don't want the fixed font (<PRE> mode), you must specify the output document type, such as text/html explicitly. This simple default behaviour is called ProVIEW-Web Calculator mode.

"dynamic" – ProVIEW web sends **NO** "Content-type" header. It is up to the script being executed to output this string. It must be of the format: "**Content-type: type/sub-type\n\n"** (i.e. **print "Content-type: text/html\n\n"**). The blank line is essential for separating the header from the document data.

type/sub-type – ProVIEW-Web will send the Content-type header using the specified type/sub-type. (i.e. text/plain, or image/jpeg)

M_cgiview Image file output format when using **view** command in ProVIEW-Web

Math Related

M_matherrflag Selects the manner in which math errors or exceptions are reported and handled. If **M_matherrflag** = 0, no

errors are reported. If **M_matherrflag** = 1, mathematical exceptions are reported and script file execution halted until you acknowledge by clicking in a popup window. If **M_matherrflag** is set equal to 2, mathematical exceptions are reported and script file execution is halted.

M_NaN holds the representation of an IEEE NaN.

Interpolation Related

M_interp Indicates which interpolation method is preferred: zero-order nterpolation (**M_interp** = 0), or linear/bilinear interpolation (**M_interp** = 1).

M_no_data Blinterp will not interpolate if an intensity value in an image is set to M_no_data.

Data Loading Related

M_maxdim This variable is used by the reada($fname,"all") call. In particular when loading raster data using the 'all' mode, it determines the maximum dimension of the array that the input image is loaded into. This is particularly usefull when loading large raster images, e.g. 10,000 x 5,000 pixels.

M_no_data Blinterp will not interpolate if an intensity value in an image is set to M_no_data.

m_ Array Window Operators

Syntax: **m_xxxx**(a)

Say 'x' is an image, then image.m_x0 is an image attribute.

A detail list of all the static attributes can be found in the section: Intrinsic Attributes Associated with Array Variables, see page 18.

See **include imgedit.msh** which allows the user to change all image attributes at one time from a table. Also, under the Image menu one can find the Edit Image Attributes item which calls imgedit.msh.

makecmplx converts a real array to complex

Syntax: **makecmplx(**x**)**

Description: converts real array to complex form

Example: The following demonstrates the operation.

```
[ready]: y = makecmplx(5)
[ready]: y
row 0 =
  5 + 0i
```

maskofEQ mask of elemens where a==b

Syntax: **maskofEQ(a,b)**

Description: return image mask with a value of one for all the pixels where a(j,i)==b(j,i). All of the maskofXX functions, e..g maskofGE(), maskofLE(), …, only accept real input arrays.

Example: The following code demonstrates maskofGE(), maskofGT(), maskofLT(), and maskofLE().

maskofBitPlane mask of bit plane n

Syntax: **maskofBitPlane(a,N,n)**

Description: return image mask with value of 0 or 1. The values will hold the bit plane 'n'. It is assumed that the input array 'a' is holding unsiged integer values coming from an integer with Nth bits. Say N = 2, then value highest value representable is 3, and bit plane n=0 corresponds to the least significant.

This function is usefull to extract encoded mask planes into integer arrays. For example one bit plane could be storing a land mask, and another cloud masks.

---

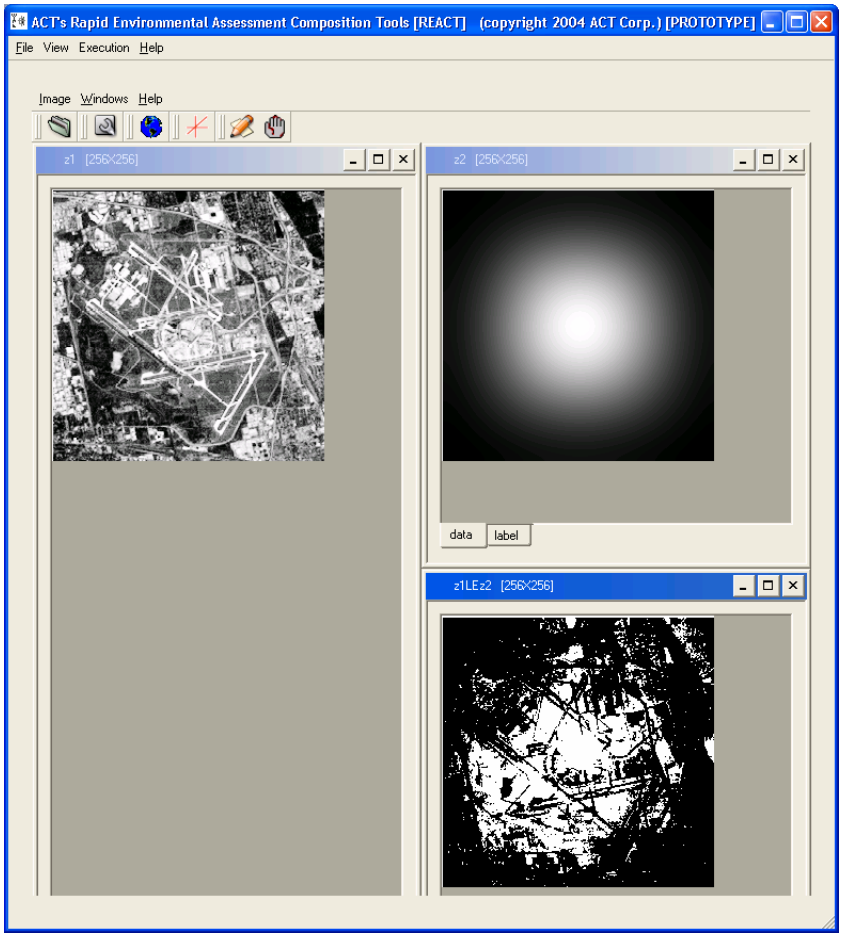
maskofGE mask of elemens where a>=b

Syntax: **maskofGE(a,b)**

Description: return image mask with a value of one for all the pixels where a(j,i)>=b(j,i). All of the maskofXX functions, e..g maskofGE(), maskofLE(), …, only accept real input arrays.

Example: The following code demonstrates maskofGE(), maskofGT(), maskofLT(), and maskofLE().

```
free all
y = hammiw(256,256)   // test pattern image
view255 y;
show y

// tests array against a constast
le=maskofLE(y,0.5)
view255 le

lt = maskofLT(y,0.5)
view255 lt

gt=maskofGT(y,0.9);
view255 gt

ge=maskofGE(y,0.9);
view255 ge
wtile
pause(-1);
free all

// test and array against another array
z1 = reada(M_proviewdir::"/images/eqohare.chr","char");
z2 = hammiw(z1.nrows,z1.ncols)*255
view255 z1
view255 z2
z1LEz2 = maskofLE(z1,z2)
view255 z1LEz2
```



---

maskofGT *mask of elemens where a>b*

Syntax: **maskofGT(a,b)**

Description: return image mask with a value of one for all the pixels where a(j,i)> b(j,i)

---

maskofLE *mask of elemens where a<=b*

Syntax: **maskofLE(a,b)**

Description: return image mask with a value of one for all the pixels where a(j,i)<=b(j,i)

---

maskofLT *mask of elemens where a<b*

Syntax: **maskofLT(a,b)**

Description: return image mask with a value of one for all the pixels where a(j,i)<b(j,i)

max *Maximum in Array*

Syntax: **max**(a) or a.**max**

Description: Find the maximum value of all the elements in the input array. Note that this function is only valid for real input arrays.

Example: The following demonstrates the operation.

```
[ready]: a = randu(1,8)
[ready]: a
row 0 =
  0.02   0.46   0.64   0.96   0.34   0.57   0.08   0.60
[ready]: max(a)
    0.95636
```

maxmin *Max and Min values in Array*

Syntax: **maxmin**(a) or a.**maxmin**

Description: Find the maximum and minimum values of all the elements in the input array. Note that this function is only valid for real input arrays.

Example: The following demonstrates the process.

```
[ready]: a = randu(1,100) //create a vector of random numbers
[ready]: maxmin(a)
row 0 =
  1.00   0.01
```

maxminx *Max&Min values in Array with exclusion*

Syntax: **maxminx**(a,exclude)

Description: Find the maximum and minimum values of all the elements in the input array, after excluding all the values in array 'exclude'. Note that this function is only valid for real input arrays.

Example: The following demonstrates the process.

```
[ready]: a = randu(100,100)
[ready]: a(0,0)= 200
[ready]: maxminx(a,200)
row 0 =
  1.00   0.01
```

maxof *Element by Element Maximum*

Syntax: **maxof**(a,b)

Description: Computes the maximum on an element by element basis. In general, the two input arrays must have the same dimensions. The only exception to this is when one of the input arrays is a scalar. Note that this function is only valid for real input arrays.

Example: The following demonstrates the process.

```
[ready]: x = randu(2,2)
[ready]: x
row 0 =
  0.01   0.22
row 1 =
  0.12   0.38
[ready]: y = randu(2,2)
[ready]: y
row 0 =
  0.48   0.02
row 1 =
  0.56   0.73
[ready]: maxof(x,y)
row 0 =
  0.48   0.22
row 1 =
  0.56   0.73
```

maxr *Row Maximum*

Syntax: **maxr**(a)

Description: Computes the maximum for each row of array "a" and strores these values as a column vector.

mbox Text Box

Syntax: **mbox(**$string**)**

Description: Prints to string to a dialog box on the screen.

mean Mean of Array Elements

Syntax: **mean**(a) or a.**mean**

Description: This module computes the mean of all the elements in the input array. The formula used to compute the mean is

$$c_{0,0} = \frac{1}{I \cdot J} \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} a_{j,i} \quad \text{for all j, i}$$

where j and i are row and column indices respectively.

Example: The following demonstrates the application.

```
[ready]: a = randu(2,100) //create a matrix of ran. num.
[ready]: mean(a)
    0.503304
```

median Compute Median Value

Syntax: **median**(a)

Description: Compute the median value of all the elements in the input array. The output is a scalar, i.e., a 1x1 array.

Example: The following demonstrates the process.

```
[ready]: a = randu(2,100)
[ready]: median(a)
    0.502454
```

medianr Row Wise Median

Syntax: **medianr(**a) or a.**medianr**

Description: Compute the median value along each row of the input array. The output is a column vector with the number of elements equal to the number of rows in the input array 'a'.

Example: The following demonstrates the process.

```
[ready]: a = randu(7,100)
[ready]: medianr(a)'
row 0 =
  0.46   0.53   0.47   0.48   0.47   0.53   0.48
```

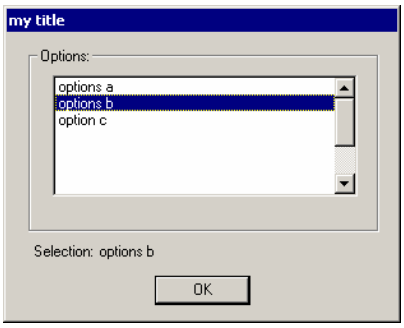
menusel *creates a Menu List Box*

Syntax: **menusel(**$title, $options**)**

Description: creates a menu list box, using the $title and $options provided to the function. Every row in $options is treated as a different option. Execution is stopped until the user selects an option and hits OK. The function returns with the row number of the option selected.

Example:

```
item = menusel("my title","options a\noptions b\noption c")
```



meter *Displays a Metering Toolbar*

Syntax: **meter**($string, value)

Description: Draws a meter toolbar on the lower right hand corner of the screen. The first argument, '$string' is a text string used for description purposes. The second argumet, 'value', is a scaler whose value controls the behavior of the meter toolbar. If 'value' < 0 the meter toolbar is erased, if 0 <= 'value' <= 100 the meter toolbar is drawn with 'value' as a percent. If 'value' > 100 an error occurs.

Example: The following script file draws a meter toolbar.

```
i=0
while(i<100){
    i = i+1
    meter("hello",i);
}
meter("hello",-1)
```

min *Minimum in Array*

Syntax: **min**(a) or a.**min**

Description: Find the minimum value of all the elements in the input array. Note that this function is only valid for real input arrays.

Example: The following demonstrates the process.

```
[ready]: a = (-2,2,1,2)
row 0 =
 -2.00  -1.00   0.00   1.00   2.00
row 1 =
 -2.00  -1.00   0.00   1.00   2.00
[ready]: min(a)
    -2
```

minof *Element by Element Minimum*

Syntax: **minof**(a,b)

Description: Compute the minimum on an element by element basis. In general, the two input arrays must have the same dimensions. The only exception to this is when one of the input arrays is a scalar. Note that this function is only valid for real input arrays.

Example: The following demonstrates the process.

```
[ready]: b = (-4,4,2,2)
[ready]: b
row 0 =
 -4.00  -2.00   0.00   2.00   4.00
row 1 =
 -4.00  -2.00   0.00   2.00   4.00
[ready]: a = (0,4,1,2)
[ready]: a
row 0 =
  0.00   1.00   2.00   3.00   4.00
row 1 =
  0.00   1.00   2.00   3.00   4.00
[ready]: minof(a,b)
row 0 =
 -4.00  -2.00   0.00   2.00   4.00
row 1 =
 -4.00  -2.00   0.00   2.00   4.00
```

minr Row Minimum

Syntax: **minr**(a)

Description: Computes the minimum for each row of array "a" and strores these values as a column vector.

mjd2dt Convert Modified Julian Date

Syntax: **mjd2dt(**mjd_value)

Description: Converts a **Modified Julian Date** to a vector giving year,month,day, hour, minute, second.

Example: The following MSHELL statement will find the date vector given a julian day number (of a given year)

```
[ready]: jdoy = 10   // find date for the 10th day of 2002
[ready]: jy = dt2mjd(2002::1::1::0::0::0)
[ready]: dt = mjd2dt(jy+ jdoy -1)
[ready]: dt
(10^0) X
row 0 =
  2002.0000      1.0000     10.0000      0.0000      0.0000
0.0000
```

momentr Row Wise Moment

Syntax: **momentr**(a,x)

Description: Compute the x^{th} moment along each row of the input array, where 'x' can be real or complex. The moment of the j^{th} row is defined as:

$$\sum_{i=0}^{a.ncols-1} a_{j,i}^{x}$$

Example: The MSHELL statements on the following page will compute the second moment of each row of the array 'a' and assign the values to 'b'.

```
[ready]: a = randu(100,5)
[ready]: b = momentr(a,2)
[ready]: show a
       --- a ---
data type is      : real
number of rows    = 100
number of columns = 5
maximum value     = 0.995382
minimum value     = 0.00161857
[ready]: show b
       --- b ---
data type is      : real
number of rows    = 1
number of columns = 5
maximum value     = 7.25416
minimum value     = 5.92651
[ready]: b
  6.48   7.25   7.18   6.13   5.93
```

mshift_u16 Bitwise shifting (16-bits unsigned integer)

Syntax: **mshift_u16**(values, nbits, left_or_right)

Description: Compute the bitwise shifting considering "values" as an ARRAY of 16-bits unsigned integers. "nbits" is the number of bits to shift and "left_or_right" is the direction (0 for left shifting, 1 for right shifting). It returns the shifted ARRAY or a NULL ARRAY in case of an error (the text attribute contains the reason).

Example:

```
[ready]: a = 1::2#3::4
[ready]: a
     1.00000,     2.00000
     3.00000,     4.00000

[ready]: mshift_u16(a, 1, 0)
     2.00000,     4.00000
     6.00000,     8.00000

[ready]:
```

mshift_u32 Bitwise shifting (32-bits unsigned integer)

Syntax: **mshift_u32**(values, nbits, left_or_right)

Description: Compute the bitwise shifting considering “values” as an ARRAY of 32-bits unsigned integers. “nbits” is the number of bits to shift and “left_or_right” is the direction (0 for left shifting, 1 for right shifting). It returns the shifted ARRAY or a NULL ARRAY in case of an error (the text attribute contains the reason).

Example:

```
[ready]: a = 1::2#3::4
[ready]: a
     1.00000,     2.00000
     3.00000,     4.00000

[ready]: mshift_u32(a, 1, 0)
     2.00000,     4.00000
     6.00000,     8.00000

[ready]:
```

Mtype Returns type of array

Syntax: **Mtype**(a)

Description: Returns zero if a is a real array, and one if it is a complex array.

- N -

nan_index *Index of all NaN elements*

Syntax: **nan_index**(a)

ncaddvar *NetCDF Add Variable*

Syntax: **ncaddvar**($fname,$varname,$vardims)

Description: Adds a new variable to an existing NetCDF file. The variable must use existing dimensions from the NetCDF file, and specify them in order in $vardims .

Example: The following demonstrates the process. The example NetCDF file has 3 dimensions: plane, lat, and lon.

```
[ready]: status =
ncaddvar("temp.nc","mydata","plane\nlat\nlon\n"
[ready]: status = ncaddvar("temp.nc","planeinfo","plane\n")
```

ncinq *NetCDF File Inquiry*

Syntax: **ncinq($fname)**

Description: Returns size and shape of NetCDF file $fname in a 1x 4 array, and a list of the variable names in the file. The format of the array is: Number of dimensions:: Number of variables:: Number of Global attributes:: The unlimited Dimension ID (if any). These numbers come directly from call to NetCDF library function 'nc_inq_var'. The .text portion of the return variable will contain a list of the variable names from the file, separated by newlines .

Example: The following demonstrates the process. The example NetCDF file has 3 dimensions, 4 variables, no global attributes, and no variables with an unlimited dimension.

```
[ready]: netcdfinfo = ncinq("temp.nc")
[ready]: netcdfinfo
(10^0) X
row 0 =
  3.00   4.00   0.00  -1.00
[ready]: netcdfinfo.text
Longitude
Latitude
Depth
temperature
```

ncinqvar NetCDF Variable Inquiry

Syntax: **ncinqvar($fname,$varname)**

Description: Returns the size and shape of variable **$varname** within NetCDF file **$fname** in a 1xN array, where N is the number of dimensions of the requested variable. The format of the array is: Size of dimension#1::Size of dimension#2::Size of dimension#3::…::Size of dimension#N. The text portion of the return variable will contain a list of the N dimension names the variable makes use of.

Example: The following demonstrates the process.

```
[ready]: varinfo = ncinqvar("temp.nc","Temperature")
[ready]: varinfo
(10^0) X
row 0 =
34.00 51.00 51.00
[ready]: varinfo.text
Depth
Latitude
Longitude
```

ncnew Create new NetCDF file

Syntax: **ncnew**($fname,$dims,varinfo,$varname)

Description: Creates a new NetCDF file. Parameter **$fname** is the filename to use. The parameter **$dims** should contain the list of dimension names for the new file separated by newlines. The **varinfo** parameter should match the **$dims** variable and provide the size for each respective dimension. Finally, **$varname** should contain the NetCDF variable name to create in the new file. Currently, MSHELL will only create a single variable NetCDF file.

Example: The following demonstrates the process. A new file called newfile.nc is created with two dimensions called dim1, and dim2. A 100x100 variable using these two dimensions is created called myvariable.

```
[ready]:
ncnew("newfile.nc","dim1\ndim2",100::100,"myvariable")
0
```

ncols Number of Columns

Syntax: **ncols**(a) or a.**ncols**

Description: Returns the number of columns in the input array.

Example: The following demonstrates the process.

```
[ready]: a = (0,255,1,10356) create 10357x256 matrix
[ready]: ncols(a)
    256
[ready]: a.ncols          // this is an equivalent command
256
```

ncread NetCDF Variable reader

Syntax: **ncread**($fname, $varname) or **ncread**($fname,$varname,start,edges)

Description: Returns the data from a NetCDF variable **$varname** from **$fname**. The first form reads in the entire variable. The second form allows an N-dimensional region of interest to be read from the variable. The **start** variable gives the start position for each dimension of the variable. The **edges** variable gives the length of each of the dimension to read.

Example: The following demonstrates the process. The MSHELL variable **img** will contain 51 rows and 51 columns from the first "plane" of the variable "Temperature". MSHELL arrays storing more than one "plane" will be filled sequentially with each plane requested.

```
[ready]: img =
ncread("temp.nc","Temperature",0::0::0,1::51::51)
[ready]: show img
--- img ---
data type is      : real
number of rows    = 51
number of columns = 51
maximum value     = 28.833
minimum value     = -1e+034
```

ncvarattstr NetCDF Variable Attribute String

Syntax: **ncvarattstr**($fname,$varname,$attname)

Description: Returns the contents of the attribute **$attname** attached to variable **$varname** from NetCDF file **$fname** as a string. An error will occur if the variable does not exist. A blank string will be returned if the attribute does not exist.

Example: The following demonstrates the process.

```
[ready]: ncvarattstr("temp.nc","Temperature","units")
degC
```

ncvarattval NetCDF Variable Attribute Value

Syntax:

Description: Returns the value of the attribute **$attname** attached to variable **$varname** from NetCDF file **$fname** as an array. An error will occur if the variable does not exist. A blank string will be returned if the attribute does not exist. Returns a Null string if attribute contains an unknown numeric type.

Example: The following demonstrates the process.

```
[ready]: ncvarattval("temp.nc","Temperature","dtgymd")
19981012
```

ncwrite NetCDF Write Variable

Syntax: **ncwrite**($fname,$varname,img,ncstart,ncedges)

Description: Writes array **img** to an existing variable **$varname** in NetCDF file **$fname**. The parameters **ncstart** and **ncedges** determine what position to start writing in the variable, and how much to write in each dimension. For an N dimensional NetCDF variable, ncstart and ncedges must be 1xN.

Example: The following demonstrates the process. The values 1::2::3 are written to the beginning of the first row in the 2-D variable "myvariable" in NetCDF file "temp.nc".

```
[ready]: ncwrite("newfile.nc","myvariable",1::2::3,0::0,1::3)
0
```

ncwriteatt NetCDF Write Attribute

Syntax: **ncwriteatt**($fname,$varname,$attname,value)or
ncwriteatt($fname,$varname,$attname,$strvalue)

Description: Creates or modifies the value of attribute $attname associated with variable $varname. If $varname is empty string, the attribute is added to the "global" attributes of the NetCDF file. A floating point value (value), or a string ($strvalue) can be written.

.

Example: The following will demonstrates the process.

```
[ready]: ncwriteatt("newfile.nc","myvariable","version",3)
0
[ready]:
ncwriteatt("newfile.nc","myvariable","Units","degrees C")
```

neqindex Non-Equality Index

Syntax: **neqindex(a,b)**

Description: Finds the locations of all the elements in an input array which are not equal to a constant value, where 'a' is the input array and 'b' is the constant scalar quantity. This function returns a (1 x M) complex array, where M is the number of points not equal to the specified value and whose array elements contain the coordinates of each point encoded as follows: the real part contains the column index, and the imaginary. part contains row index of the point. If all elements are matched, a null value is returned.

Example: The following MSHELL statement will compute the….

```
[ready]: moe = 1#2#3
[ready]: neqindex(moe,1)
row0 =   0.00+  1.00i   0.00+  2.00i
```

nint *Nearest Integer*

Syntax: **nint**(a) or a.**nint**

Description: Compute the nearest integer for each element in the array.

Example: The following MSHELL statement will compute the nearest integer for each element of the input array, ‘a’, and store the result in ‘c’.

```
[ready]: a = randu(2,4)   // create a random 2x4 matrix
[ready]: nint(a)
row 0 =
  0.00   1.00   1.00   0.00
row 1 =
  1.00   1.00   0.00   0.00
```

nlines *Returns number of Lines*

Syntax: **nlines**($string)

Description: This function returns the number of lines contained in the string "$string".

nrows *Number of Rows*

Syntax: **nrows**(a) or a.**nrows**

Description: Returns the number of rows in the input array.

Example: The following demonstrates the process.

```
[ready]: a = (0,255,1,10356)  //create 10357x256 matrix
[ready]: nrows(a)
    10356
[ready]: a.nrows          // this command is equivalent
    10356
```

- O -

ones *Initialize an Array to all ones*

Syntax: **ones**(n,m)

Description: Create an array of the specified dimensions, with all elements set equal to 1.

Example: The following MSHELL statement will create a 512 x 512 array with all entries set to 1,

```
[ready]:a = ones(512,512)
[ready]: a(300,0:4)   // print first 5 elements in row 301
row 0 =
  1.00   1.00   1.00   1.00   1.00
```

openf *Open a file for Formatted I/O*

Syntax: **openf**(unit,"fname","mode")

Description: This function is used to open a disk file for formatted input or output. The unit number can be any positive integer value. The selected integer value will identify the opened file from then on. The file name, “fname” must be a valid file name in DOS. The “mode” string must be one of the followings: "r" for read; "w" for write; or "a" for append.

If the write or append modes are tried on a non-existing file, the file will be created. The **openf** returns a 0 into status if successful. The system variable **M_path** will determine which directories, in addition to the current directory, to be searched.

If ‘fname’ contains any environment variables in the form ‘$(VAR)’, those are expanded to their values (see “expand” command).

See Also: **writef** and **closef**

Example: The following will open "test.out" for output as unit number 1, and "test.in" for input as unit number 2,

```
[ready]: status = openf(1,"test.out","w");
[ready]: status = openf(2,"test.in","r");
```

- P -

pause *Suspend Execution for N Seconds*

Syntax: **pause**(N)

Description: Will causes MSHELL to stop execution for 'N' seconds. If the value of 'N' is negative a pause dialog box will be opened. Note that 'N' can be a fractional number of seconds.

pdsfindobj *Get a list of objects within a PDS file*

Syntax: **pdsfindobj**($fname)

Description: Returns the name of each PDS object found in $fname. Object examples would be IMAGE, or BROWSE_IMAGE.

pdskwdstr *Retrieve a PDS keyword value*

Syntax: **pdskwdstr**($label_string, $kwd_name)

Description: Returns the value of the PDS keyword specified in $kwd_name if found in the complete PDS label $label_string.

Example: The following MSHELL code will retrieve two keywords from a PDS file.

```
[ready]: eqohare =
reada("C:/PROVIEW/IMAGES/eqohare.pds","pds")
[ready]: $hdr = eqohare.text
[ready]: pdskwdstr($hdr, "PDS_VERSION_ID")
PDS3
[ready]: pdskwdstr($hdr, "IMAGE/LINES")
256
```

pixval *Displays Pixel Status of Mouse*

Syntax: **pixval**(a)

Description: This displays the pixel value and intensity corresponding with the placement of the mouse when clicked within an image "a" ,after having entered this command at the command line window.

Example: pixval(x)

Now click in the image "x" where you would like placement and intensity status.

The following is returned:

<row position> <column position> <intensity> <button value>

The <button value> is 1 for a left click or 2 for a right click. Also, if the selection is made outside of the image, then the first three values are all -1.

---

plot Plot a Vector

Syntax: **plot**[#](y) or **plot**[#](x,y)

Description: Plots a row vector. The parameter '#' in the plot function is an optional integer (0 to 255) that selects the plot screen where the plot will be placed. If an integer number from 0 to 255 is provided in this field, the generated plot can be indexed from then on by that number. For example if 'x' is a row vector, then **plot**10(x); will plot the vector on plot screen number 10. If you would like to free that screen later on you can type **free plot**10. Note that the plot function can have either one or two arguments.

Single Argument Case: When one argument is used in the plot function as in **plot**(y), the row vector 'y' will be plotted against an internally generated ramp of integer values. If 'y' is a complex vector then the real part of 'y' will be used as the abscissa and the imaginary part of 'y' as the ordinate.

Two Argument Case: When two arguments are used with the plot function, as in **plot**(x,y); the first argument, 'x' , corresponds to the abscissa values and the second argument, 'y', corresponds to the ordinate values, i.e. a plot of 'y' versus 'x' will result. Hence, for the case that 'y' is complex, **plot**(y), is equivalent to **plot**(**real**(y), **imag**(y)).

See Also: "**M_"** for a complete list of system variables which affect the plot function.

Example: The following MSHELL instructions generate two ramp vectors in memory, 'x' and 'y', and plot **sin**(y) as a function of **cos**(x); see Figure below and code following.

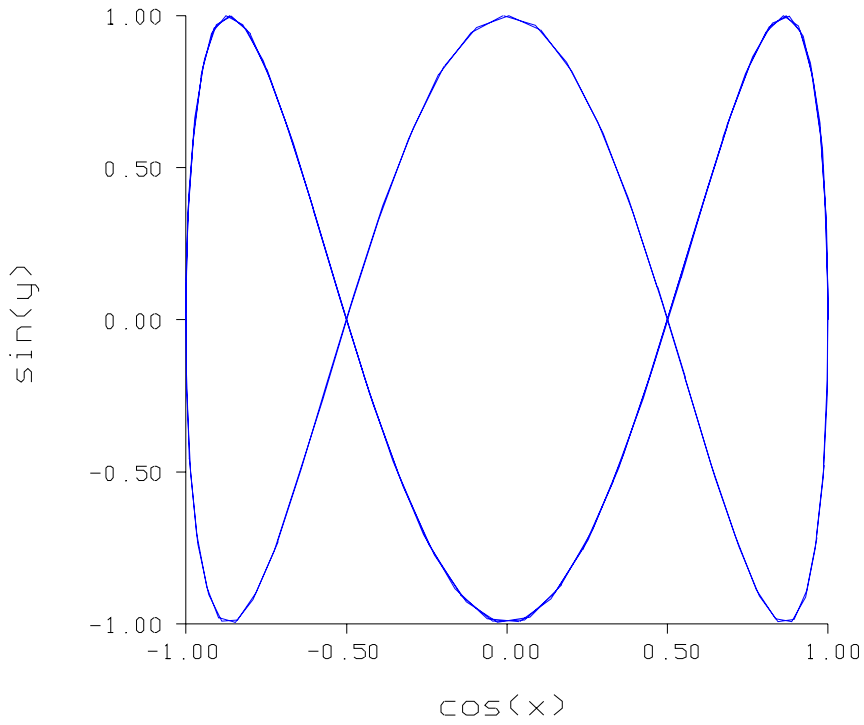


Figure 5

```
[ready]: x = (0,3.14159*5,0.1);
[ready]: y = 3.0 * x ;
[ready]: M_ylabel = "sin(y)";
[ready]: M_xlabel = "cos(x)"
[ready]: plot(sin(y),cos(x));
```

plot3d 3-D Plot or Mesh Plot

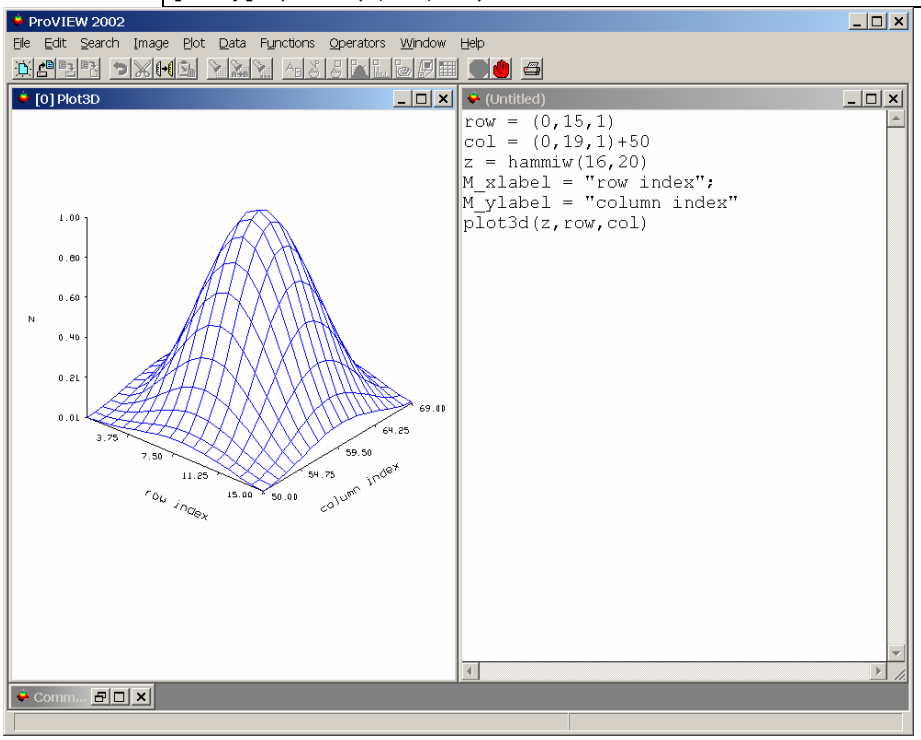
Syntax: **plot3d**[#](z)

Description: This function generates a hidden line surface plot using the values of an input array, **'z'.** The **plot3d** function requires as arguments a real array, 'z', with two or more rows, and two or more columns. You can also provide a row and column axis vector for annotation. The parameter '#' in the plot function is an optional integer (0 to 255) that selects the plot screen where the plot will be placed. If an integer number from 0 to 255 is provided in this field, the generated plot can be indexed from then on by that number For example if 'x' is an array, then **plot3d**10(x) will plot 'x' on plot screen number 10. If you want to free that screen later on you can type **free plot**10.

See Also: "M_ for a complete list of system variables which affect the plot functions.

Example: The following MSHELL instructions generate a 32 x 32 Hamming function and displays the function as a 3d plot.

```
[ready]: row = (0,15,1)
[ready]: col = (0,19,1)+50
[ready]: z = hammiw(16,20)
[ready]: M_xlabel = "row index";
[ready]: M_ylabel = "column index"
[ready]: plot3d(z,row,col)
```



polyfill *Fills an Image with Polygons*

Syntax: **list = polyfill(image, complex(value,fill)::polylist)**

Description: Fills and Image with Contained Polygons. This routine allows for the fast drawing of polygons on '**image**'. '**polylist**' is a complex array where each row correspond to a polygon. Each element in polylist is a complex number with row and column position. The column number is encoded as the real part and the row number as the imaginary part.
Image. 'value' is a column vector containing the real amplitude values to assign to the polygon to be drawn. 'fill is a flag vector that determines if the polygon will be drawn or not. The values of fill must be zero or one.

.

Example: The following MSHELL expression illustrates the use of the polyfill function**.**

```
fill      = ones(3,1)*1
value     = (255::128::64)'
image     = zeros(256,256)
polylist= complex(0::100::150::0,0::0::100::100)

polylistm = (polylist#(polylist*1.5)#(polylist*2))
polylistm = rmirror(polylistm)

control = complex(value,fill)::polylistm
out       = polyfill(image,control )

view      image
out
```

polygonorient Computes the orientation of a polygon

Syntax: result = polygonorient(poly)

Description: This function computes the orientation of a polygon, given the coordinates of its vertices. The coordinates are specified as a complex row vector. Only closed polygons are supported, so the last vertex should be the same as the first one (see the example below.)

Return values:

0 The given polygon is degenerate

1 Means Clock-Wise orientation (CW)

-1 Means Counter-Clock-Wise orientation (CCW)

Example:

```
// clock-wise polygon:

a=complex(0,0)::complex(0,10)::complex(10,10)::complex(10,0):
:complex(0,0)

s = polygonorient(a)

s // s == 1
```

preg_match Regular Expression Matching

Syntax: preg_match($str_to_search, $regular_expression_pattern)

Description: Given a regular expression pattern, this function finds **the first match** in the input string.

General documentation on using regular expressions can be found here:
http://www.perldoc.com/perl5.8.0/pod/perlre.html

See also preg_replace

Example: The following MSHELL expression illustrates the use of the regulars expression capabilities. Given the following code,

```
$a = "moe\nTEMP_MAX,IMG_MAX,IMG_MIN,IMG_MEAN,MOE,LARRY,CURLY"
b = preg_match($a,"IMG_([^,]+),")

print "input string:\n"::$a::"\n"

print "\noutput of preg_match, i.e. b.text\n"::b.text::"\n"

print "\noutput of preg_match, i.e.
b\n"::float2str(b,"%.0f")::"\n"
```

once executed it generates,

```
input string:
moe
TEMP_MAX,IMG_MAX,IMG_MIN,IMG_MEAN,MOE,LARRY,CURLY

output of preg_match, i.e. b.text
IMG_MAX,
MAX


output of preg_match, i.e. b
13,21
17,20
```

Row 0 of b contains the start and end position of the first occurrence of the pattern. Subsequent rows will contain any “captured substrings” called out in the pattern with ()’s.

Note: end position is returned as one character beyond the match found, a convenient start position value to use to continue search of rest of input string.

Also, b.text will contain the all the substrings specified in b, one per line.

The following provides additional examples. By no means this is a complete set of examples.

```
$a =
"moe\ntesting\nTEMP_MAX,IMG_MAX,IMG_MIN,IMG_MEAN,MOE,LARRY,LA
RBA,TEEEMP_MAX,CURLY"

print "\n\n"

print "-----------------EXAMPLE of Using Anchors ------------
--------\n"
print "Note: Anchor are used to limit a match to a particualr
location in the string, \n"
print "       ^   for the start of the string\n"
print "       $   for the end of the string\n"


print "a '^'  at the beginning of a string means that it must
match the beginning of the string\n"
$pattern  = "^mo"
b= preg_match($a,$pattern)
print "INPUT STRING:\n"::$a::"\n"
print "SEARCH PATTERN is: "::$pattern::"\n"
print "\nOUTPUT of preg_match, i.e. b.text\n"::b.text::"\n"
print "\nOUTPUT of preg_match, i.e.
b\n"::float2str(b,"%.0f")::"\n"

print "Note: a '$'  at the end of the regular expression
string means that it must match the end of the string\n"
$pattern = "(?m)ing$" // use (?m) to activate inline option
to use ^ to match after and  $ to match before internal new
lines
b= preg_match($a,$pattern)
print "INPUT STRING:\n"::$a::"\n"
print "SEARCH PATTERN is: "::$pattern::"\n"
print "\nOUTPUT of preg_match, i.e. b.text\n"::b.text::"\n"
print "\nOUTPUT of preg_match, i.e.
b\n"::float2str(b,"%.0f")::"\n"



print "-----------------EXAMPLE of character classes --------
------------\n"
print "Note: all characters within a [] are treated as a set
of acceptable characters\n"
$pattern = "LAR[AB]"
b= preg_match($a,$pattern)
print "INPUT STRING:\n"::$a::"\n"
print "SEARCH PATTERN is: "::$pattern::"\n"
print "\nOUTPUT of preg_match, i.e. b.text\n"::b.text::"\n"
print "\nOUTPUT of preg_match, i.e.
b\n"::float2str(b,"%.0f")::"\n"
$pattern = "M[A-E]+N" // allow in the class  A or B or C or D
or E
b= preg_match($a,$pattern)
print "INPUT STRING:\n"::$a::"\n"
```

```
print "SEARCH PATTERN is: "::$pattern::"\n"
print "\nOUTPUT of preg_match, i.e. b.text\n"::b.text::"\n"
print "\nOUTPUT of preg_match, i.e.
b\n"::float2str(b,"%.0f")::"\n"

print "-----------------EXAMPLE of repeating sequences ------
-------------\n"
print "Note: repeating sequences are specified by the regular
expression quantifiers\n, i.e.\n"
print "       ?   specifies 0 or 1 \n"
print "       *   specifies 0 or more \n"
print "       +   specifies 1 or more\n"
print "       {n} specifies exactly n times\n"
print "       {n,m}   specifies at least n, no more than m
times \n"
print "       {n,}    specifies at least n times \n"
$pattern = "LAR[AB]+"
b= preg_match($a,$pattern)
print "INPUT STRING:\n"::$a::"\n"
print "SEARCH PATTERN is: "::$pattern::"\n"
print "\nOUTPUT of preg_match, i.e. b.text\n"::b.text::"\n"
print "\nOUTPUT of preg_match, i.e.
b\n"::float2str(b,"%.0f")::"\n"

$pattern = "T(E{3,})MP"
b= preg_match($a,$pattern)
print "INPUT STRING:\n"::$a::"\n"
print "SEARCH PATTERN is: "::$pattern::"\n"
print "\nOUTPUT of preg_match, i.e. b.text\n"::b.text::"\n"
print "\nOUTPUT of preg_match, i.e.
b\n"::float2str(b,"%.0f")::"\n"


$pattern = "IMG_([^,]+),"
// the above pattern is looking for IMG_ followed by one or
more non-commas characters and then a comma
b = preg_match($a,$pattern)
print "INPUT STRING:\n"::$a::"\n"
print "SEARCH PATTERN is: "::$pattern::"\n"
print "\nOUTPUT of preg_match, i.e. b.text\n"::b.text::"\n"
print "\nOUTPUT of preg_match, i.e.
b\n"::float2str(b,"%.0f")::"\n"

print "Note: a '$'  at the end of the regular expression
string means that it must match the end of the string\n"
$pattern = "(?m)^.*ing$"  // use (?m) to activate inline
option to use ^ to match after and  $ to match before
internal new lines
b= preg_match($a,$pattern)
print "INPUT STRING:\n"::$a::"\n"
print "SEARCH PATTERN is: "::$pattern::"\n"
print "\nOUTPUT of preg_match, i.e. b.text\n"::b.text::"\n"
```

```
print "\nOUTPUT of preg_match, i.e.
b\n"::float2str(b,"%.0f")::"\n"




print "Note: a '$'  at the end of the regular expression
string means that it must match the end of the string\n"
$a = "Hi, I am 44 years old, and I can do 45 push-ups in 3
seconds."
$pattern = "([0-9]+) years old.*?([0-9]+) push-ups.*?([0-9]+)
seconds"  // use (?m) to activate inline option to use ^ to
match after and  $ to match before internal new lines
b= preg_match($a,$pattern)
print "INPUT STRING:\n"::$a::"\n"
print "SEARCH PATTERN is: "::$pattern::"\n"
print "\nOUTPUT of preg_match, i.e. b.text\n"::b.text::"\n"
print "\nOUTPUT of preg_match, i.e.
b\n"::float2str(b,"%.0f")::"\n"
```

preg_replace Regular Expression Replace

Syntax: **preg_replace**($str_to_search, $regular_expression_pattern, $replacement_expression)

Description: Regular expression string replace support

Example: The following MSHELL expression illustrates the use of the regulars expression capabilities

```
$a = "Fred Flintstone\nBarney Rubble\nWilma Flintstone\nBetty
Rubble\n"
"----"
$a
"----"
$c = preg_replace($a,"(\\w)\\w* (\\w+)","$1 $2")
$c
"----"



----
Fred Flintstone
Barney Rubble
Wilma Flintstone
Betty Rubble

----
F Flintstone
B Rubble
W Flintstone
B Rubble

----
```

General documentation on using regular expressions can be found here:
http://www.perldoc.com/perl5.8.0/pod/perlre.html

These functions are based on the "Perl Compatible Regular Expressions" library (PCRE). Information on PCRE can be found at: **http://www.pcre.org/**. The topics in the PCRE manual pages on usage of regular expressions within the C API are applicable to the MSHELL functions which make use of them.

---

print *Formatted Print*

Syntax: **print** "..."

Description: Prints scalar values or strings to the standard output.

Example: The following MSHELL expression illustrates the use of the print statement

```
[ready]: M_format = "000.000000"
[ready]: x = 3
[ready]: print "The value of x = " x "\n"
The value of x = 3.000000
```

putenv Assign to sys environment variable

Syntax: **putenv**($assignment)

Description: Assigns an operating system environment variable given an assignment expression in $assignment.

Example: The following MSHELL statement will assign an environment variable

```
[ready]: putenv("MYVAR=value")
[ready]: getenv("MYVAR")
value
```

- Q -

qgauss *Area Under Gaussian Density*

Syntax: **qgauss**(a) or a.**qgauss**

Description: Let Z(x) be the one dimensional Gaussian density function with zero mean and unit variance, defined as

$$Z(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2},$$

then **qgauss** returns the integral of Z(x) from x to infinity,

$$\int_{a_{j,i}}^{\infty} Z(x)dx$$

for each element in the input array.

Example: The following MSHELL expressions compute the area under the Gaussian density function (the probability) for values of 0 to 5 standard deviations.

```
[ready]: sigma = (0,5,0.1)
[ready]: plot(sigma,qgauss(sigma))
```

qgaussinv *Inverse of qgauss*

Syntax: **qgaussinv**(a) or a.**qgaussinv**

Description: Given an input array with values, with 0.0 < $a_{j,i}$ < 1.0 that correspond to probabilities under the normalized Gaussian density function, **qgaussinv** computes the decision threshold values, $t_{j,i}$, that will generate those probability values. That is, **qgaussinv** computes the values of $t_{j,i}$ such that the area under the normalized Gaussian density between $t_{j,i}$ and infinity equals $a_{j,i}$, i.e.

$$a_{j,i} = \int_{t_{j,i}}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx$$

Example: The following MSHELL expression computes the ordinate values under the Gaussian density function for a row vector with values of 0.25, 0.50, and 0.75.

```
[ready]: qgaussinv( 0.25::0.50::0.75 )
  0.67   0.00  -0.67
```

- R -

randg Gaussian Random Number Generator

Syntax: **randg**(n,m)

Description: This function is similar to **randu**, except that it generates independent identically distributed (i.i.d.) Gaussian random numbers with zero mean and unit variance, where, 'n' and 'm' refer to the number of rows and columns in the array of Gaussian random numbers to be generated.

Example: Say, 'a' is an already existing array to which it is desired to add Gaussian distributed random numbers with zero mean and standard deviation of 10. The following MSHELL expression will add the Gaussian noise to 'a' and will save the result in c,

```
[ready]: randinit(30) //initialize random number generator
[ready]: a = (10,100,1)
[ready]: c = a+10. * randg( a.nrows, a.ncols);
[ready]: plot(a,c)
```

randinit Random Number Seed Initializer

Syntax: **randinit**(n)

Description: As this function is used to initialize the MSHELL random number generators it does not return any value. To initialize a random sequence, call **randinit** using a scalar value greater than 1.

Example: The initialization and subsequent use of a random generator,

```
[ready]: randinit(10)     // initialize random number
generators
[ready]: x = randg(1,100) // call gaussian. rand. num. gnrtr
```

randu Uniform Random Number Generator

Syntax: **randu**(n, m)

Description: This generator produces independent identically distributed (i.i.d.) random numbers between 0 and 1. It returns a pointer to an (n x m) dimensioned array structure, where n and m are, respectively, the number of rows and columns of uniformly distributed random numbers generated.

The uniform random number generator itself is an adaptation of the **RAN1( )** generator presented in "Numerical Recipes in C", see page 3, "**References and Further Readings**". This is a portable generator (given the same seed number it will generate the same random sequence on all machines), which utilizes three linear congruential generators in producing output.

Example: The following MSHELL expressions will add uniform noise, with intensities between 0 and 10, to the array, 'a', and save the result in 'c',

```
[ready]: randinit(20) //initialize random number generator
[ready]: c = a+10. * randu( nrows(a) , ncols(a) )
```

rcoeff Correlation Coefficient

Syntax: **rcoeff**(a,b)

Description: Estimates the correlation coefficient between two arrays, 'a' and 'b' with the same dimensions. The correlation coefficient is a scalar. Although the correlation coefficient is an internal function,. it could also be computed in terms of other MSHELL instructions as,

```
rcoeff(a,b) = (mean(a*.b)-a.mean*b.mean )/sqrt(a.var*b.var) .
```

react_get_carto_info Retrieve Cartographic Viewer information

Syntax: **react_get_carto_info**($info_type)

Description: Retrieve Cartographic Viewer information depending on the specified information type string ("$info_type"). If the function call is successful, it returns an ARRAY containing the requested information (both as ARRAY elements and text attribute). In case of an error, it returns a NULL ARRAY and the text attribute contains the error description.

The following "$info_type" strings are allowed:

- "basic"; the returned ARRAY is a row vector ARRAY containing: *carto_UL* (complex value representing the upper left corner in cartographic coordinates of the current view in the Cartographic Viewer), *carto_dxy* (complex value representing the x and y meters-per-pixel values of the current view in the Cartographic Viewer), *carto_Nxy* (complex value representing the current view width and height in the Cartographic Viewer); the text attribute is set to the map projection string of the current view in the Cartographic Viewer.

- Empty string (same as "basic")

react_set_carto_map_info Set Cartographic Viewer Map Info

Syntax: **react_set_carto_map_info**($map_info)

Description: Set the Cartographic Viewer Map Info to the specified string. It returns 0 upon successful, 1 otherwise (text attribute will contain the error description).

react_load_mlt Loads an MLT file into REACT

Syntax: result = react_load_mlt($file_mlt, mode)

Description: Loads the specified Map Layers Tree file (.mlt) into the REACT's Layers Manager. The "mode" argument controls how the mlt has to be added:

mode == 0: replace current tree

mode == 1: append to current tree

mode == 2: prepend to current tree

The returned value will be 0 on success; otherwise an error code is returned, along with a description of the error in the ".text" attribute.

Example:

```
$file_mlt = "my_layers.mlt"
result = react_load_mlt( $file_mlt, 0 )

if ( result ) {

          // error...
          result
          result.text

}
```

react_update_layers *Updates Layers visualization*

Syntax: result = react_update_layers()

Description: Updates the Cartographic Viewer with the visualization of the current layers tree loaded in the Layers Manager.

The returned value will be 0 on success; otherwise an error code is returned, along with a description of the error in the ".text" attribute.

Example:

```
result = react_update_layers()

if ( result ) {

        // error...
        result
        result.text
}
```

react_enable_view Shows a REACT panel

Syntax: result = react_enable_view($panel)

Description: Makes sure that the specified REACT panel is visible. Currently these are the supported panels:

V_LAYERS (Layers Rendering Tree)

V_3D (3D Viewer)

V_SITES (Target Panel)

V_ANALISYS (Development Environment)

The returned value will be 0 on success; otherwise an error code is returned, along with a description of the error in the ".text" attribute.

Example:

```
result = react_enable_view( "V_LAYERS" )

if ( result ) {

        // error...
        result
        result.text
}
```

react_open_url Opens an URL in a browser window

Syntax: s = react_open_url($my_url)

Description: This function opens the specified URL in a web browser window. The functionality is available only in GUI mode, i.e. within REACT.

Example:

```
$url = "http://www.actgate.com/"
s = react_view_url( $url )
```

react_get_property Get the value of a REACT property

Syntax: result = react_get_property($prop_name)

Description: This function returns the value of the specified REACT property. REACT has a hierarchical set of objects, and those objects have properties.

In order to access those properties, a "dot" notation is used. For example, "obj1.obj2.obj3.propname" retrieves the "propname" property value of the specified object "obj3" following the REACT objects hierarchy "obj1.obj2.obj3".

A status value (zero = successful, non-zero = failure) is returned. Its text attribute will contain the command output (upon successful) or a brief description of the error (upon failure).

Example:

```
result=react_get_property("react.mdev.monitoringPanelDocked")

if ( result ) {

        // error...
        result
        result.text

} else {

        $value = result.text
}
```

react_get_properties Get the values for a set of REACT properties

Syntax: result = react_get_properties($react_object)

Description: This function returns all the property values of the specified object (following the REACT objects hierarchy) as an XML string. If an empty string is specified (""), it retrieves the entire collection of all properties values of all REACT objects as an XML string.

A status value (zero = successful, non-zero = failure) is returned. Its text attribute will contain the command output (upon successful) or a brief description of the error (upon failure).

Example:

```
result = react_get_properties( "react.ncda" )

if ( result ) {

        // error...
        result
        result.text

} else {

        $xml = result.text
}
```

The output XML string looks like this:

```
<Object name="react">
        <Object name="ncda">
                <Property name="activeTab">layers</Property>
        </Object>
</Object>
```

---

react_set_property Set the value of a REACT property

Syntax: result = react_set_property($prop_name, $prop_value)

Description: This function changes the property value of the specified object (following the REACT objects hierarchy) to the specified value.

A status value (zero = successful, non-zero = failure) is returned. Its text attribute will contain the command output (upon successful) or a brief description of the error (upon failure).

Example:

```
result=react_set_property("react.mdev.monitoringPanelDocked",
"1")

if ( result ) {
```

```
        // error...
        result
        result.text

} else {

        // successful
}
```

reada Read Array or Image

Syntax: **reada**("fname",$mode[, cntvec])

Description: Reads an array or image from disk. The '$mode' string specifies the format of the data to be read from the disk. If the wrong mode is used with a given file the image will not load properly. The image file formats (or modes) supported are summarize in the following table, followed by a more in-depth description of earch format mode.

If 'fname' contains any environment variables in the form '$(VAR)', those are expanded to their values (see "expand" command).

See Also: Importing Data **, page 36.**

Table 1 Summary of Read-Array/ Writea-Array Format Modes

Format Mode	Brief Description	Bytes/pixel	Read support in MSHELL	Write Support in MSHELL
all	Tries to figure out if the input image is char, gif, float, bmp, jpeg, … or any other format. At this time it does not cover all the possible formats supported by reada(). This format make use of the system variable M_maxdim which		Yes	N/A

	determines the maximum image dimension to use.			
char	MSHELL simple format with a 9 byte header	1 byte/pixel	Yes	Yes
flex	MSHELL reader with powerfull region of interest reader	Multiple …	Yes	N/A
Fits	Flexible Image Transfer format	IEEE float	Yes	Yes, simple format
Tiff	Industry Standard	N/A	Yes	Yes, see also writecolor() function
Gif	Industry Standard	N/A	Yes	Yes
pds	Planetary Data System Support (basic format used in planetary missions)		Yes	Yes (simple format)
float	MSHELL simple format with a 9 byte header	4 bytes /pixel	Yes	Yes
double	MSHELL simple format with a 9 byte header	8 bytes / pixel		
bmp	Industry standard format	N/A	Yes	Yes
jpeg	Industry standard format	N/A	Yes	Yes
asciiflex	MSHELL flexible text loader into numeric array		Yes	N/A
clemen_pds	Clementine Mission Specific	DCT type compression	Yes	N/A

Note that only the charflex and asciiflex formats requires the additional argument ‘cntvec’, please see examples.

Note: for the 'char', 'float', 'ascii', and 'double' formats, reada will attempt to load the x.text attribute if it was intitialized in the input file.

The 'char' internal format is good to store and retrieve 1 byte data

The **char** format stores images by using 1 byte/pixel prefixed by MSHELL's 9 byte header, i.e. 4 bytes specifying the number of rows, 4 bytes specifying the number of columns, and 1 byte specifying if array elements real or complex. This format can be used both for both the reading and writing of data.

Example: Read **eqohare.chr** image using 'chr' format.

```
[ready]:  x = reada("eqohare.chr","char");
[ready]:  view x
```

The 'flex'format can be used to read almost any image that can be described with the flex-cntvec structure.

The **flex** format provides you significant flexibility when reading various byte/pixel types of data. You can read the whole image or just specified subregions of the image. This is accomplished by using an additional argument, 'cntvec'. This is a vector with specific data read criteria, and is formated as follows:

```
cntvec =                                                    \\
in_sizej::    /* number of rows in input image */           \\
in_sizei::    /* number of col. in input image */           \\
hsize::       /* header size in bytes to skip */            \\
jstart::      /* start row */                               \\
istart::      /* start column */                  \\
jend::        /* end row */                                 \\
iend::        /* end column */                              \\
jstep::       /* read every 'jstep' row */                  \\
istep         /* read every 'istep' column */
```

The syntax for this format is then: **reada**("eqohare.chr","flex_type",cntvec)

The format specifier is "flex_type"; this string can be any of the following:

"char"

"PC16"

"PC32"

"SUN16"

"SUN32"

"PCFLOAT"

"SUNFLOAT"

"PCDOUBLE"

Example: Reading a portion of the **eqohare.chr** image using 'flex'

```
[ready]: eqohare=reada("eqohare.chr","char", \\
                  256::256::9::128::128::255::255::3::1)
[ready]: view eqohare
```

This format is easier to use from the menu with the **File|Image Open** option since a dialog box, illustrated in Figure 4, is provided to facilitate the input of the data read criteria.

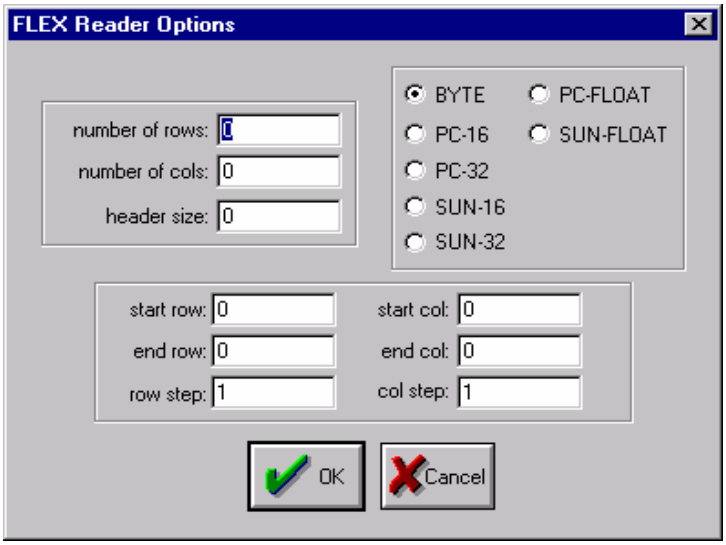


Figure 4 - Charflex dialog box

The 'float' format is a MSHELL internal format for storing single floating point arrays (real or complex)

The **float** format is similar to the char format but the data is stored in floating point using 4 bytes/pixel.

Example: Creating, writing, and then reading in a file in **float** format,

```
[ready]:  y = writea("out.flt", randu(3,3), "float")
[ready]:  x = reada("out.flt","float")
```

The 'double' format is a MSHELL internal format for storing single double precision point arrays (real

The **double** format is similar to the char format but the data is stored in double precision using 8 bytes/pixel.

or complex)

Example: Creating, writing, and then reading in a file in **double** format,

```
[ready]:  y = writea("out.dbl", randu(3,3), "double")
[ready]:  x = reada("out.dbl","double")
```

The 'ascii' format is a MSHELL internal format for storing arrays as ASCII values

The **ascii** format stores images in ASCII. A typical ASCII file will look like

```
   3     3    0
-1       0    1
-2       0    2
-1       0    1
```

where the first row is a three element header containing the number of rows, the number of columns, and the real-complex flag (0 = real, 1 = complex). This header is followed by the image data stored lexico-graphically in ASCII. This format can be used for both the reading and writing of data.

Example: Creating, writing, and then reading in a file in **ascii** format,

```
[ready]: y = writea("out.asc", randu(3,3), "ascii")
[ready]: x = reada("out.asc","ascii")
```

asciiflex

The **asciiflex** format permits you to load selected rows and columns from an ascii file. Note that the elements in the ascii file must be comma delimited. A typical ASCII(flex) could look like **test.asc,** below:

```
       This is a sample user file
 3,  1,  gain=  2, tint=3
-1,  0,  gain= 22, tint=3
-2,  1,  gain=n/a, tint=3
 0,  2,  gain=  3, tint=0
This is the end of the file
```

where ascii text can be mixed in with the numerical data to be extracted. Note that this read process searches for and accepts the first number it encounters after a delimiter. The syntax is similar to that of char(flex) , using a control vector, 'cntvec" to provide the specific read parameters. The control vector structure, which is differs from that of charflex, is formated as follows:

```
       first_row::   /* first row to read */ \\
last_row::    /* last row, if -1 read to last row */ \\
nnchar::      /* non numeric character flag */ \\
hdrsubset::   /* vector of column numbers to read */ \\
cntvec = firstrow::lastrow::nnchar::colvec
cntvec.text = ","
```

The text portion of the control vector contains the delimiter to use for ascii reading. Note that the non-numeric charecter flag value, 'nnchar' is assigned to any array entry encountered in the read process

that does not contain an ascii numeric data. The syntax for this format is then:

reada("test.asc","asciiflex",cntvec)

Example: Reading a portion of the **test.asc** file.

```
[ready]: cntvec = 1::-1::-999::0::2 //read rows 1 to
end,columns 0 and 2

[ready]: cntvec.text = ","

[ready]: data = reada("test.asc","asciiflex",cntvec)
 [ready]: data
 (10^0) X
row 0 =
  3.00   2.00
row 1 =
 -1.00  22.00
row 2 =
 -2.00 -999.00
row 3 =
  0.00   3.00
row 4 =
-999.00 -999.00
```

Note that in the above example, only the numeric data was extracted from the array entries of column 2; in those cases, rows 3 and 5, that contained non numeric data the designated 'nnchar' value has bee assigned. This flexability in extraction of numeric information from ascii text commented data arrays is a very powerful processing tool.

BMP

The **bmp** format is the windows Bit Map Format (BMP). This format can be used for both the reading and writing of images.

Example: Reading a file in **bmp** format,

```
[ready]: x = reada("mandel.bmp","bmp")
```

PDS

The **PDS** format used is the implementation of the Planetary Data System Format adopted by the Clementine mission.

Within MSHELL there are other functions to support the reading of PDS type data, see internal functions ‘pdsfindobj’ and ‘pdswdstr’.

Example: Reading a file in **PDS** format,

```
[ready]:  // reads image object
[ready]:  x = reada("lua1472h.202","clemen_pds_image_uncorr")
[ready]:  // reads browse image object
[ready]:  x=reada("lua1472h.202","clemen_pds_browse_uncorr")
[ready]:  // reads histogram object
[ready]:  x=reada("lua1472h.202","clemen_pds_hist_uncorr")
```

See Also: **readtext(), readtext_and_status(), readf(), writea(), openf(), closef(), writecolor()**

readf *Formatted File Read*

Syntax: **readf**(unit,"format",arrayname) or

readf(unit,"format",stringname)

Description: This function is used to perform a formatted read of a scalar or a string from a file unit which has already been opened using the **openf** command. Only one value or string can be read at a time. The format string follows a similar form to the "C" language formatting options, where %s is used for strings, and %f, %g, %e are used for floating point numbers. The first command above is used for reading a value into an already existing array. After executing the **readf** the array will have dimensions of 1x1, i.e. a scalar. The second command above is used for reading a string into an already existing string variable. If the **readf** is successful it will return a value of 0. On end-of-file (eof) a value of -1 is returned.

See Also: **openf an M_**.

Example: Assuming that unit 1 has been already open with an **openf** statement, and that the variables 'x' and 'name' are respectively, an array variable and a string variable, then the following MSHELL statements are legal **readf** statements,

```
[ready]: status = readf(1,"%s",$name);
[ready]: status = readf(1,"%f",x);
[ready]: status = readf(1,"%4.2f",x);
[ready]: status = readf(1,"%g",x);
[ready]: status = readf(1,"%e",x); .
```

readtext Loads Text File

Syntax: **readtext**($fname)

Description: This function is used to load a text file into a memory string. If '$fname' contains any environment variables in the form '$(VAR)', those are expanded to their values (see "expand" command).

Example:

```
// load a file into a memory string
$x    =
readtext(M_proviewdir::"/scripts/filtering/spatial_ops/gradie
nt_ops/gradient.msf")
print $x  // print the content of the string

// load a URL into a string
$x    = readtext("http://www.actgate.com")
```

readtext_and_status Loads Text File with check

Syntax: **readtext_and_status**($fname)

Description: This function is used to load a text file. It returns a status variable (see example code). If the value is zero the operation was a success. Similar to readtext(), it can be used also to read URLs.

If '$fname' contains any environment variables in the form '$(VAR)', those are expanded to their values (see "expand" command).

Example: ,

```
$file =
M_proviewdir::"/scripts/filtering/spatial_ops/gradient_ops/gr
adient.msf"
status = readtext_and_status($file)
if(status==0){
    print $x  // print the content of the string
}
```

real Real Part of an Array

Syntax: **real**(a) or a.**real**

Description: Returns real part for each element in the array 'a'.

Example: The following MSHELL statement will extract the real part of the Fourier Transform of 'a' , and store the result in 'c',

```
c = real( fft2(a) );
```

```
[ready]: evaltext $x
```

regionand Defines a ROI as Common of Two

Syntax: **regionand**(roi,roi2)

Description: Defines a new region of interest which is the common (overlapping) area of two other regions of interest (roi **and** roi2).

return Returns From an Include File

Syntax: **return**

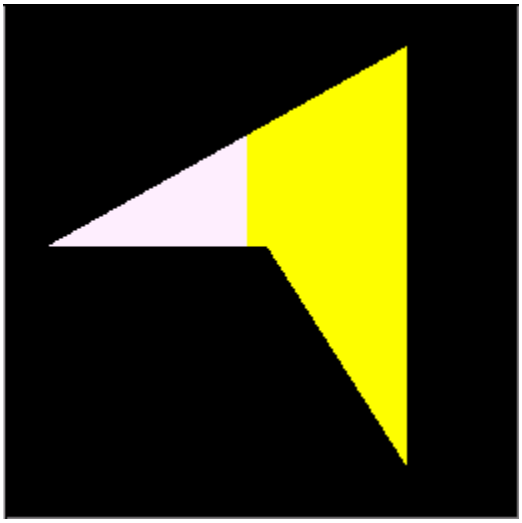
Description: Stops the execution of an include file and returns control to the calling process.

rfill Fills a subregion

Syntax: **rfill**(a,b)

Description: Finds the subregion of an image**(**'b'**)** which is all points contained within a created groi from a list of polygonal vertices**(**'a'**)**.

Example:



```
[ready]: groi = complex(20,120)::complex(200,20)::complex(200,230)::complex(130,120)
[ready]: x = zeros(255,255)
[ready]: groi2 = rfill(groi,wdef(0,0,255,255))
[ready]: x(groi2) = 100
[ready]: groi3 = rfill(groi,wdef(0,0,120,120))
[ready]: x(groi3) = 200
[ready]: _
```

rindex Range Index

Syntax: **rindex**(a,minval,maxval)

Description: This function finds the location of all the elements of an input array which **fall within** a specified range of values, where 'a' is the input array, and 'minval' and 'maxval' are scalar values specifying the selection range. This function returns a (1 x M) complex array, where M is the number of points equal to the specified value and whose array elements contain the coordinates of each point encoded as follows: the real part contains the column index, and the imaginary. part contains row index of the point. If no elements are matched, the value **-1** is returned.

Example: The following illustrates the function using the array 'a'.

```
[ready]: a = randu(2,6)   // create random 2x6 matrix
[ready]: a
row 0 =
  0.08   0.59   0.23   0.46   0.93   0.13
row 1 =
  0.79   0.19   0.39   0.33   0.71   0.67
[ready]: rindex(a,.4,.5)
3 + 0i
```

rindexc Range Index Complement

Syntax: **rindexc**(a,minval,maxval)

Description: This function finds the location of all the elements of an input array which fall **outside** within a specified range of values, where 'a' is the input array, and 'minval' and 'maxval' are scalar values specifying the selection range. This function returns a (1 x M) complex array, where M is the number of points equal to the specified value and whose array elements contain the coordinates of each point encoded as follows: the real part contains the column index, and the imaginary. part contains row index of the point. If no elements are matched, the value **-1** is returned.

rmirror Row Mirror

Syntax: **rmirror**(a) or a.**rmirror**

Description: This function yields the mirror image of the input array over its rows, e.g., the last row will become the first row.

See Also: **cmirror**

Example: Creates array 'a' and then does the row reflection,

```
[ready]: a = randu(3,3)
[ready]: a
row 0 =
  0.96   0.42   0.93
row 1 =
  0.17   0.31   0.90
row 2 =
  0.75   0.58   0.65
[ready]: rmirror(a)
row 0 =
  0.75   0.58   0.65
row 1 =
  0.17   0.31   0.90
row 2 =
  0.96   0.42   0.93
```

rmshell_execute Execute a command in the remote MSHELL

Syntax: **rmshell_execute**($handle, $command)

Description: This function works after an rmshell_session_init and can be used to execute a command in the remote MSHELL interpreter. It takes two arguments: a valid string handle to a previously opened remote MSHELL interpreter and a command string to be executed. This function will block until the execution is finished. If successful, it returns 0 and the text attribute holds the output string

generated by the execution of $command. If an error occurs, it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

Example: see rmshell_getvar example.

rmshell_execute_async Execute a command in the remote MSHELL (does not block)

Syntax: **rmshell_execute_async**($handle, $command)

Description: This function works after an rmshell_session_init and behaves essentially like the rmshell_execute function but it returns immediately. It takes two arguments: a valid string handle to a previously opened remote MSHELL interpreter and a command string to be executed. If successful, it returns 0, otherwise it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

rmshell_wait Wait the number of specified seconds

Syntax: **rmshell_wait**($handle, n_sec)

Description: This function works after an rmshell_session_init and an rmshell_execute_async. It takes two arguments: a valid string handle to a previously opened remote MSHELL interpreter and the number of seconds to wait. If it is –1, this function blocks until the remote server is finished executing the command. Otherwise it waits the number of seconds specified. If the remote server has not yet finished executing the command when the number of seconds elapsed or if an error occurs, it returns a non-zero value, otherwise it returns 0 and the text attribute holds the output string generating by the execution of $command.

rmshell_getfile Download a file from the remote MSHELL

Syntax: **rmshell_getfile**($handle, $filelist)

Description: This function works after an rmshell_session_init and can be used to download a file from the remote MSHELL interpreter. It takes tow arguments: a valid string handle to a previously opened remote MSHELL interpreter and a comma-separated string containing the input file name and the output file name. If successful, it returns 0, otherwise it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

Example: see rw_getInventory example.

rmshell_getvar Get an ARRAY variable from the remote MSHELL

Syntax: **rmshell_getvar**($handle, $varname)

Description: This function works after an rmshell_session_init and can be used to retrieve the content of an ARRAY variable defined in the remote MSHELL interpreter. It takes tow arguments: a valid string handle to a previously opened remote MSHELL interpreter and the name of the ARRAY variable we want to retrieve the content. If the specified variable does not exist or if an error occurs, it returns a NULL array and the text attribute can be used to retrieve a brief description of the error.

Example: The following example shows most of the remote MSHELL functions being utilized on a simple test case.

```
// opening Remote MSHELL session
$srvr = rmshell_session_init( “212.48.10.150” )
if ($srvr = “”) {
          print “Error opening Remote MSHELL session”
          return
}

x = hammiw(256, 256)
$s = “I like pineapples”

// whenever ‘status’ is a non zero-value, its text attribute
// can be used to retrive a brief description of the error
status = rmshell_putvar($srvr, “x”)
if (status) {
          status.text
          return
}
status = rmshell_putvarS($srvr, “$s”)

// performing operations remotely
status = rmshell_execute($srvr, “x = x.scale255”)
status = rmshell_execute($srvr, “$s = smodify($s, \”pine\”,
\”\”)”)

// getting back results
x = rmshell_getvar($srvr, “x”)
$s = rmshell_getvarS($srvr, “$s”)

view x
$s

// closing Remote MSHELL session
status = rmshell_session_close($srvr)
```

rmshell_getvarS Get a string variable from the remote MSHELL

Syntax: **rmshell_getvarS**($handle, $varname)

Description: This function works after an rmshell_session_init and can be used to retrieve the content of a string variable defined in the remote MSHELL interpreter. It takes tow arguments: a valid string handle to a previously opened remote MSHELL interpreter and the name of the string variable we want to retrieve the content. If the specified variable does not exist or if a an error occurs, it returns an empty string.

Example: see rmshell_getvar example.

rmshell_putfile Upload a file from the remote MSHELL

Syntax: **rmshell_getfile**($handle, $filelist)

Description: This function works after an rmshell_session_init and can be used to upload a file to the remote MSHELL interpreter. It takes tow arguments: a valid string handle to a previously opened remote MSHELL interpreter and a comma-separated string containing the input file name and the output file name. If successful, it returns 0, otherwise it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

rmshell_putvar Put an ARRAY variable to the remote MSHELL

Syntax: **rmshell_putvar**($handle, $varname)

Description: This function works after an rmshell_session_init and can be used to put the content of an ARRAY variable in the remote MSHELL interpreter. It takes tow arguments: a valid string handle to a previously opened remote MSHELL interpreter and the name of the ARRAY variable we want to put the content. If successful, it returns 0, otherwise it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

Example: see rmshell_getvar example.

rmshell_putvarS Put a string variable to the remote MSHELL

Syntax: **rmshell_putvarS**($handle, $varname)

Description: This function works after an rmshell_session_init and can be used to put the content of a string variable in the remote MSHELL interpreter. It takes tow arguments: a valid string handle to a previously opened remote MSHELL interpreter and the name of the string variable we want to put the content. If successful, it returns 0, otherwise it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

rmshell_session_close Close connection to a remote MSHELL interpreter

Syntax: **rmshell_session_close**($IP_address)

Description: This function works after an rmshell_session_init and can be used to close a connection to a remote MSHELL interpreter. It takes a valid string handle to a previously opened remote MSHELL interpreter. If successful, it returns 0, otherwise it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

Example: see rmshell_getvar example.

---

rmshell_session_init Connect to a remote MSHELL interpreter

Syntax: **rmshell_session_init**($handle)

Description: This function is used to connect to a remote MSHELL interpreter. It takes the IP address (as a string) of the remote machine where the MSHELL interpreter is running. If successful it returns a valid string handle, otherwise it returns en empty string.

Example: see rmshell_getvar example.

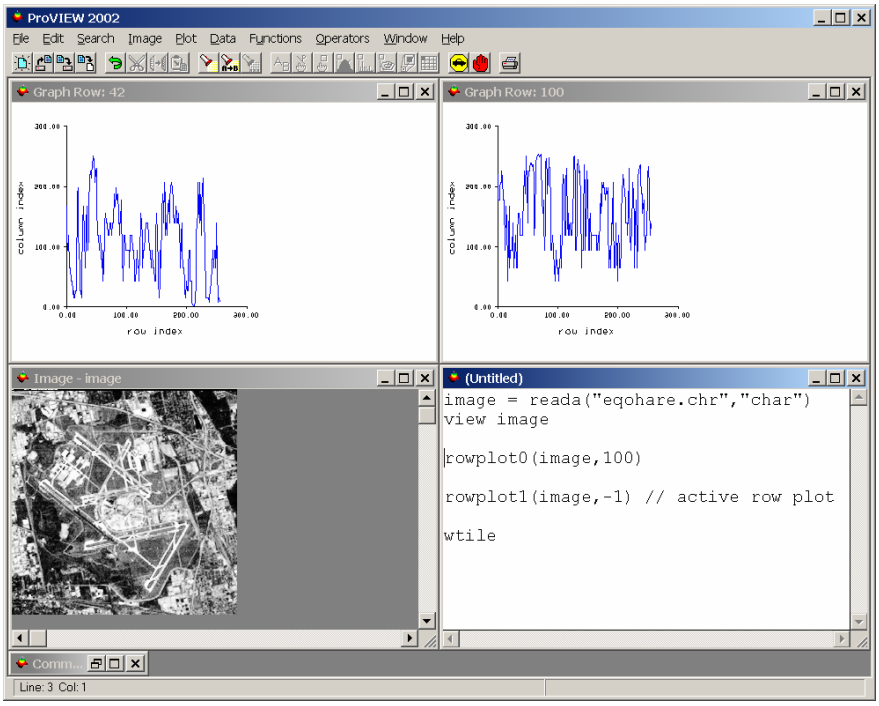
---

rowplot Plots a Row from Array

Syntax: rowplot[N](array,row#)

Rowplot[N](array,-1) ← for interactive plots

Description: Used to plot a particular row from an array.



rs_availWipeServers Returns available WIPE Server.

Syntax: **$WipeServers = rs_availWipeServers(rs1)**

Description: This function returns a list of names of available WIPE Servers.

Example:

```
[ready]: $WipeServers = rs_availWipeServers(rs1)
[ready]:
```

rs_availDataSets Returns list of available data sets.

Syntax: **rs_AvailDataSets(rs1,"ACTTest")**

Description: This function returns a list of available data sets in a particular WIPE Server.

Example:

```
[ready]: $DataSets = rs_AvailDataSets(rs1,"ACTTest")
[ready]:
```

rs_extractData extract data from the Wipe Server.

Syntax: **rs_extractData(rs1,**

Description: This function is used to extract data from the WIPE Server. The input arguments are the remote server handle, the WIPE Server name, the data item id you wish to extract data from, the layer number, the ROI (subregion), the output image dimensions, the format to save the data in and the filename to save the data into. If the ROI has all the values set to 9999.0, it means we want the whole region the data item occupies. If this function succeds, the return value is 1, otherwise is 0.

Example:

```
[ready]: $RecId = " TEDS_3Dtemp|nwlan_1999071200_tfin"
[ready]: bbox = ones(1,4)*9999.0
[ready]: dim = 41::181
[ready]: layer = 0
[ready]: $format = "netCDF"
[ready]: $fname = "d:\\temp\\temp.nc"
[ready]:ret=rs_extractData(rs1,"ACTTest",$RecId,layer,bbox,di
m,$format,$fname)
```

rs_getMetaData Extract the metadata request .

Syntax: **rs_getMetaData(,,,,,,,)**

Description: This function takes as parameters the remote server handle, the name of the WIPE Server, a string containing a list of data set names (comma separated), a string containing the field names to extract from the SQL table, the ROI (ullon::ullat::lrlon::lrlat), start time and end time. The output is a string containing the metadata requested for all the data items in the SQL table that meets the ROI and TOI constraints.

Example:

```
[ready]:$MetaData=rs_getMetaData(rs1,"ACTTest","Sar,TEDS_3Dte
mp","DataSet,ID",-180::90::180::-90,"","")
[ready]:
```

rs_init loads msh32rs.dll

Syntax: **rs_init()**

Description: This function loads (dynamically) msh32rs.dll. This DLL contains all the CORBA client code to connect and invoke methods on a remote WIPE Object server . If it succeeds it returns 0, otherwise –1.

Example:

```
[ready]: status = rs_init()
[ready]:
```

rs_connect connect to a remote WIPE Object

Syntax: **rs_connect("erie.actgate.com",12000)**

Description: This function is used to connect to a remote WIPE object. It takes two parameters: a string containing the ip address of the WIPE Object host machine, and a port number (this is where WIPE Object Factory Server is set up to listen in). If Successful it returns a handle number, otherwise it returns –1. The handle number is used as a parameter in the other functions.

Example:

```
[ready]: rs1 = rs_connect("erie.actgate.com", 12000)
[ready]:
```

---

rs_release closes connection with a WIPE Object

Syntax: **rs_release (rs1)**

Description: This function is used to close the connection with a remote WIPE Object.

Example:

```
[ready]: rs_release(rs1)
[ready]:
```

---

rs_exec Executes command in the remote server

Syntax: **rs_exec(rs1,$cmd)**

Description: This function is used to execute an MSHELL command string in the remote WIPE Object server. It returns 0 if the command executed properly on the remote machine, otherwise it returns the MTOOL error code generated during execution.

Example:

```
[ready]: $cmd = "x = scale255(hammiw(128,128))\n"
[ready]: retval = rs_exec(rs1,$cmd)
```

---

rs_exec_async Same as rs_exec (does not block)

Syntax: **rs_exec_async(rs1,$cmd)**

Description: This function is similar to rs_exec, excepts that it does not block. The local thread of execution is free to process other commands while the command passed in rs_exec_async executes in the remote server. If succesfull it returns 0.

Example:

```
[ready]: $cmd = “include \”bigloop\”\n”
[ready]: retval = rs_exec_async(rs1,$cmd)
```

rs_getString Retrieves the content of a string variable.

Syntax: **rs_getString(rs1,$varname)**

Description: This function is used to retrieve the contents of a string variable in the remote WIPE Object server. If the string variable does not exists, the function returns a NULL value.

Example:

```
[ready]: $cmd =”$name=\”Hello World!\”\n”
[ready]: retval = rs_exec(rs1,$cmd)
[ready]: $varname = “$name”
[ready]: $temp = rs_getArray(rs1,$varname)
```

rs_getArray Retrieves the content of an Array variable

Syntax: **rs_getArray(rs1,$varname)**

Description: This function is used to retrieve the content of an array variable in the remote WIPE Object server. If the array variable does not exists, the function returns a NULL array.

Example:

```
[ready]: $cmd =”x =scale255(hammiw(128,128))\n”
[ready]:retval = rs_exec(rs1,$cmd)
[ready]: $varname = “x”
[ready]: x = rs_getArray(rs1,$varname)
```

rs_wait Waits the number of specified seconds

Syntax: **rs_wait(rs1, #of seconds)**

Description: This function is used after a call to rs_exec_async. The number of seconds to wait is passed in to the function. If it is –1, this function blocks until the remote server is finished executing the command. Otherwise it waits the number of seconds specified. If the remote server has not finished executing the command when the number of seconds elapsed it returns 1, otherwise it returns 0.

Example:

```
[ready]://wait for 5 seconds; loop until server is finished
executing command
[ready]:while (rs_wait(rs1,5)){
                          "server is not finished yet… \n"
}
```

rw_getInventory Retrieve Inventory XML String

Syntax: **rw_getInventory**($handle, $request)

Description: This function works after an rmshell_session_init and can be used to perform a WIPE server data coverage request by using the remote MSHELL interpreter. It takes two arguments: a valid string handle to a previously opened remote MSHELL interpreter and the coverage XML string. This function will block until the WIPE server response is ready. If successful, it returns 0 and the text attribute holds the output inventory XML string. If an error occurs, it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

Example:

```
// opening a Remote MSHELL session
$srvr = rmshell_session_init( “212.48.10.150” )
if ($srvr = “”) {
          print “Error opening Remote MSHELL session”
          return
}

// loading data coverage request template from file
$cov = readtext("covTemplate.txt")
// performing request and checking for errors
status = rw_getInventory($srvr, $cov)
if (status) {
          status.text
          return
}

// if request has been performed successfully,
// s.text contains the XML inventory string
$inventory = s.text
$inventory

if ( !xmlparse($inventory) ) {
          print "\nErrors in XML inventory string.\n"
          return
}

$inventoryImageRelPath = xmltextvalue("CoverageImage")

// the __WROOT__ macro contains the WIPE Server root
directory string
// where the output image file is located
$fn =
"__WROOT__"::$inventoryImageRelPath::",..\\images\\inventory.
jpg"
// retrieving output image file
status = rmshell_getfile($srvr, $fn)

inventoryImage = reada("..\\images\\inventory.jpg", "all")
view inventoryImage

// closing the Remote MSHELL session
status = rmshell_session_close($srvr)
```

rw_getInventoryAsync Retrieve Inventory XML String (does not block)

Syntax: **rw_getInventoryAsync**($handle, $ request)

Description: This function works after an rmshell_session_init and behaves essentially like the rw_getInventoryXML function but it returns immediately. It takes two arguments: a valid string handle to a previously opened remote MSHELL interpreter and the coverage XML string. If successful, it returns 0,

otherwise it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

Example: see rw_waitForInventory example.

rw_getMosaic Retrieve MOSAIC XML String

Syntax: **rw_getMosaic**($handle, $ request)

Description: This function works after an rmshell_session_init and can be used to perform a WIPE server data extraction request by using the remote MSHELL interpreter. It takes two arguments: a valid string handle to a previously opened remote MSHELL interpreter and the MOSAIC request XML string. This function will block until the WIPE server response is ready. If successful, it returns 0 and the text attribute holds the output MOSAIC XML string. If an error occurs, it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

Example:

```
// opening a Remote MSHELL session
$srvr = rmshell_session_init( “212.48.10.150” )
if ($srvr = “”) {
          print “Error opening Remote MSHELL session”
          return
}

// loading data extraction request template from file
$idb = readtext("idbTemplate.txt")
// performing request and checking for errors
s = rw_getMosaic($h, $idb)
if (status) {
          status.text
          return
}

// if request has been performed successfully,
// s.text contains the XML MOSAIC string
$mosaic = s.text
$mosaic

if ( !xmlparse($mosaic) ) {
          print "\nErrors in XML mosaic string.\n"
          return
}

$mosaicImageRelPath = xmltextvalue("OutputImage")
$mosaicDataRelPath = xmltextvalue("OutputData")

// the __WROOT__ macro contains the WIPE Server root
directory string
// where the output image file is located
$fn =
"__WROOT__"::$mosaicImageRelPath::",..\\images\\mosaic.jpg"
// retrieving output image file
status = rmshell_getfile($h, $fn)

mosaicImage = reada("..\\images\\mosaic.jpg", "all")
view mosaicImage
// closing the Remote MSHELL session
status = rmshell_session_close($srvr)
```

rw_getMosaicAsync Retrieve MOSAIC XML String (does not block)

Syntax: **rw_getMosaicAsync**($handle, $ request)

Description: This function works after an rmshell_session_init and behaves essentially like the rw_getMosaicXML function but it returns immediately. It takes two arguments: a valid string handle to a previously opened remote MSHELL interpreter and the MOSAIC request XML string. If successful, it

returns 0, otherwise it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

Example: see rw_waitForMosaic example.

rw_waitForInventory Wait the number of specified seconds

Syntax: **rw_waitForInventory**($handle, n_sec)

Description: This function works after an rmshell_session_init and an rw_getInventoryXMLAsync. It takes two arguments: a valid string handle to a previously opened remote MSHELL interpreter and the number of seconds to wait. If it is –1, this function blocks until the remote server is finished executing the command. Otherwise it waits the number of seconds specified. If the remote server has not yet finished executing the data coverage request when the number of seconds elapsed or if an error occurs, it returns a non-zero value, otherwise it returns 0 and the text attribute holds the output inventory XML string..

Example:

```
// opening a Remote MSHELL session
$srvr = rmshell_session_init( “212.48.10.150” )
if ($srvr = “”) {
          print “Error opening Remote MSHELL session”
          return
}

// loading data coverage request template from file
$cov = readtext("covTemplate.txt")
nSecsToWait = 5

// performing request asynchronously and checking for errors
status = rw_getInventoryAsync($srvr, $cov)
if (status) {
          status.text
          return
}

// now waiting for the response to be ready
print "waiting "::int2str(nSecsToWait)::" seconds... "
isBusy = rw_waitForInventory($srvr, nSecsToWait)
while ( isBusy ) {
          print "server is still busy\n"
          print "waiting "::int2str(nSecsToWait)::"
seconds... "
          isBusy = rw_waitForInventory($srvr, nSecsToWait)
}
print "server is ready"

// if request has been performed successfully,
// ‘isBusy.text’ contains the XML inventory string
$inventory = isBusy.text
$inventory

// closing the Remote MSHELL session
status = rmshell_session_close($srvr)
```

rw_waitForMosaic Wait the number of specified seconds

Syntax: **rw_waitForMosaic**($handle, n_sec)

Description: This function works after an rmshell_session_init and an rw_getMosaicXMLAsync. It takes two arguments: a valid string handle to a previously opened remote MSHELL interpreter and the number of seconds to wait. If it is –1, this function blocks until the remote server is finished executing the command. Otherwise it waits the number of seconds specified. If the remote server has not yet finished executing the data extraction request when the number of seconds elapsed or if an error occurs, it returns a non-zero value, otherwise it returns 0 and the text attribute holds the output MOSAIC XML string.

Example:

```
// opening a Remote MSHELL session
$srvr = rmshell_session_init( “212.48.10.150” )
if ($srvr = “”) {
          print “Error opening Remote MSHELL session”
          return
}

// loading data coverage request template from file
$cov = readtext("idbTemplate.txt")
nSecsToWait = 5

// performing request asynchronously and checking for errors
status = rw_getMosaicXMLAsync($srvr, $idb)
if (status) {
          status.text
          return
}

// now waiting for the response to be ready
print "waiting "::int2str(nSecsToWait)::" seconds... "
isBusy = rw_waitForMosaicXML($srvr, nSecsToWait)
while ( isBusy ) {
          print "server is still busy\n"
          print "waiting "::int2str(nSecsToWait)::"
seconds... "
          isBusy = rw_waitForMosaicXML($srvr, nSecsToWait)
}
print "server is ready"

// if request has been performed successfully,
// ‘isBusy.text’ contains the XML MOSAIC string
$mosaic = isBusy.text
$mosaic

// closing the Remote MSHELL session
status = rmshell_session_close($srvr)
```

- S -

SatVIEW computes spacecraft geometry info

Description: this function has been changed to an external stand alone application. See Appendix.

save Saves Arrays or Strings to Disk

Syntax: **save $**savefile array1 array2 array3 …
save $savefile all
save $savefile all_local
save $savefile allstr

Description: Used to save arrays to disk wherby one specifies the desired path and array names to use separated by spaces.

To save all of the ARRAY variables use **save $savefile all**.

To save all the string variables use **save $savefile allstr**.

To save all local ARRAY variable use **save $savefile all_local**

See Also: **load**

scale255 Scale to 8-bit Range

Syntax: **scale255**(a) or a.**scale255**

Description: Scale the input array values to fall in the range [0,255]. This function is particularly useful for scaling an array prior to copying it to an 8 bit image frame buffer.

Note that the input to scale255 must be real.

Example: Given that 'a' is a valid array,

```
f = scale255(a);
```

will scale 'a' and copy it into image 'f'.

setcwd Sets the current working directory

Syntax: **setcwd(**$string**)**

status = setcwd($string**)**

Description: This is used to set the current working directory from the command line or within a script. The advantage of using this function over the typical M_cwd command is that this captures the state of the change. If status is <0> then the cwd request was successful; if equal to <-1> then it was unsuccessful. This could be because the directory specified does not exist or simply that access permissions have not been met.

Example: The following sets the current working directory to "C:\Temp". Since status = 0, we know that the request was successful.

```
[ready]: status = setcwd("C:\\Temp")
[ready]: status
   0
[ready]:
```

select Selects an Output Look-Up Table

Syntax: **select wolut#**

Description: This is used to select the desired look-up table. The # sign will be replaced with the corresponding number for the look-up table desired.
NOTE: THIS FUNCTION IS NOT USED BY REACT.

Example: The following will select look-up table 2.,

```
[ready] select wolut2
```

setroi Interactively sets an ROI from

Syntax: **setroi(**image**)**

Description: This is used to set a roi from the screen using the mouse for input. After having released the left mouse buton and clicking the right button, the roi will be saved as the rectangular region selected.

Example: The following sets the variable 'roi' equal to the complex array of pixels defining the dragged out contained region within 'image'.

```
[ready] roi = setroi(image)
```

shiftc Cyclic Shift of an Image

Syntax: **shiftc**(a,row,col)

Description: This function is used to shift or translate an input array or image by a specified number of columns and rows. The shift is cyclic, i.e., border pixels will wrap around. Note that the values of 'row" and 'col' must be non-negative.

Example: Included with example of **shiftt**.

shp_from_csv_ll create shapefile from ascii input file

Syntax: **status = shp_from_csv(ll**($inputCSV,$output,$prefix)

Description: This function is used to convert a CSV-formatted list of geometric features (points, open or closed polygons) to an ESRI compliant shapefiles.

The first argument is the input file, the second argument is the output directory and the third argument is a prefix to be put in the output filenames.

The input file contains ASCII text lines, with comma-separated values. The first line is the header and contains the names (again comma-separated) of the data columns. Those names need to be less or equal than 12 characters in length.
Two of the columns must be "lonv" and "latv" containing the longitude and latitude coordinates of the geometric features. If a feature has more than one vertex, the corresponding coordinates have to be separated by double colon. In case of closed polygons, the lon/lat coordinates of the first and the last vertex must coincide.

Here is an example of input file:

feat_i,	descr,	lonv,	latv
p1,	point,	140.0,	-82.5
l2,	open,	120.3::130.0	-60.6::-79.8
p2,	closed,	-180::-140::-140::-180::-180,	-82::-82::-87::-87::-82

The function returns a status code, and in case of errors a brief message is returned in the .text attribute.

Status code legend:

0	No errors
1	Input file not found
2	Cannot read input file
7	Column names > 12 characters
8	Exactly one 'lonv' and one 'latv' column must be present
21	'lonv' and 'latv' coordinates do not match in feature
22	Missing 'lonv' / 'latv' columns
9	Line fields number does not match the header
26	Cannot write feature
6	Cannot create feature
3	Shapefile driver not available
4	Creation of output file failed
5	Layers creation failed

shp2contour Shapefile to Contour Image

Syntax: **shp2contour**($filename, bbox, size, recn)

Description: This function generates an image containing a view of the shapefile for a given region and number of records.

‘**$filename**’ is a string containing the name of the shapefile file.

‘**bbox**’ is a 1x4 array containing the bounding box of the region of interest (Xmin, Ymin, Xmax, Ymax), **bbox.text** – optional input containing the ‘proj4’ string that describes the coordinate system of the vector features. Note: This option will be executed as long as there is a valid ‘.prj’ file associated with the shape file. The ‘.prj’ file can contain a WKT (well-known-text) representation of the coordiate system.

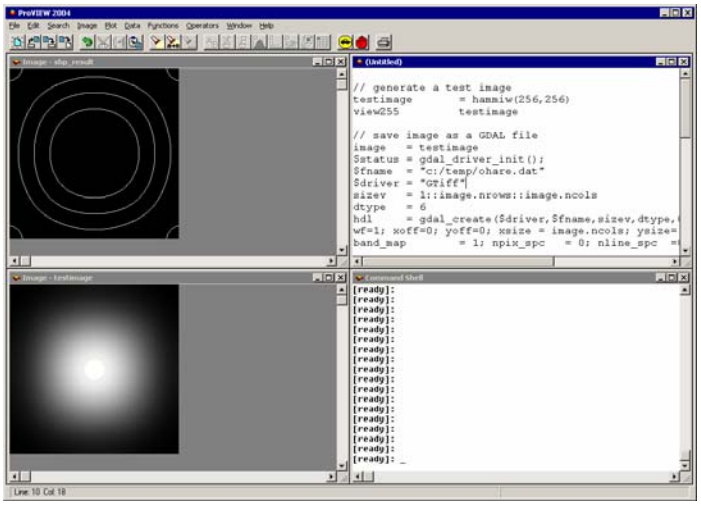
‘**size**’ is a 1x2 array containing the desired image size (rows, columns), and

‘**recn**’ is either a Nx2 array or 0. If ‘recn’ is a Nx2 array, the first column contains the shapefile record id and the second column contains the value to use when drawing the contour for that record; if it is 0, then all the shapefile records are used and the value contained in the system variable M_plotcolor is used for the contour. The generated image can be used as an overlay to another image..

Example: The following example illustrates the function.

```
// generate a test image
testimage     = hammiw(256,256)
view255   testimage

// save image as a GDAL file
image     = testimage
$status   = gdal_driver_init();
$fname    = "c:/temp/ohare.dat"
$driver   = "GTiff"
sizev     = 1::image.nrows::image.ncols
dtype     = 6
hdl       = gdal_create($driver,$fname,sizev,dtype,0)
wf=1; xoff=0; yoff=0; xsize = image.ncols; ysize=
image.nrows; my_array= image;
band_map  = 1; npix_spc   = 0; nline_spc  =0; nband_spc   =0
status    =gdal_rasterio(hdl,wf,xoff,yoff,xsize,ysize,\\
          my_array,band_map,npix_spc,nline_spc,nband_spc);

// set parameters to call gdal_contour_generate
band      =1; cinterval=-1; cbase=0
flevels=0.5::0.25::0.1::0.01
flevelcount=flevels.ncols
use_nodata=0
$shapefilename = "c:/temp/shapeout"
$attr="temperature_values"
use3d=0
status    = gdal_contour_generate(hdl,band,cinterval,\\
          cbase,flevels,use_nodata,$shapefilename,$attr,use3d
)
status=gdal_close(hdl);

/*** View created shapefile as contours ***/
M_format = "00000000000.0000"
$shpfile = $shapefilename::".shp"
ifo       = getshpinfo($shpfile)
xmin      = ifo(0,4); ymin    = ifo(0,5)
xmax      = ifo(0,6); ymax    = ifo(0,7)
bounds    = xmin::ymin::xmax::ymax
sz        = image.nrows::image.ncols; recn    = 0
shp_result= shp2contour($shpfile,bounds, sz,recn)
view      shp_result
wtile
```

See Also: gdal_contour_generate, and functions whose name start with ‘shp’, shp2image, shp2fillimage, …



shp2image Shapefile to Image

Syntax: **shp2image**($filename, image, bbox, recn, mode)

Description: This function is **useful when performing polygon fill** of shapefile records of type 5 (polygons). Note: the function shp2fillimage ends up calling this function.

'$filename' is a string containing the name of the shapefile file, 'image' is an array variable used as input to the polygon fill routine (it serves as a background image),

'bbox' is a 1x4 array containing the bounding box of the region of interest (Xmin, Ymin, Xmax, Ymax),

bbox.text – optional input containing the 'proj4' string that describes the coordinate system of the vector features. Note: This option will be executed as long as there is a valid '.prj' file associated with the shape file. The '.prj' file can contain a WKT (well-known-text) representation of the coordiate system.

'recn' can be a Nx2 array, a Nx1 array or 0, and

'mode' is a value that determines the type of output generated:

'0' – fill polygon,
'1' – fill polygon using shapefile record id as the fill value, and
'2' – region add.

The following examples show the different modes in action.

Mode 0 : Polygon Fill

This mode works exactly as shp2fillimage. If 'recn' is a Nx2 array, for each row, the first column contains the shapefile record id and the second column contains the value to use to fill in the corresponding

polygons; if it is a Nx1 array, the column vector contains the shapefile ids to extract from the shapefile; and if it is 0, then all the shapefile records are extracted. The value used to fill in the polygons in the second and third case is determined by the content of the system variable M_plotcolor. The returned image is a combination of the input image and the filled in polygons.

Example: The following example illustrates the function.

```
z = ones(180,360)*100
c = shp2image("d:\\temp\\admin98.shp", z, -180::-90::180::90,
0, 0)
view c
```

Mode 1 – Polygon Fill using Shapefile Record Id

In this mode the record id is used to fill in the polygons. If 'recn' is a Nx1 array, the column vector contains the record ids to extract from the shapefile; if 'recn' is 0 then all the records are extracted. Note that if 'recn' is a Nx2 array the second column is ignored.

Example: The following example illustrates the function.

```
z = ones(256,256)*100
c = shp2image("d:\\temp\\sed.shp", z, -78.015::33.001::-
74.992::36.009, 0, 1)
view c
show c
--- c ---
data type is      : real
number of rows    = 256
number of columns = 256
maximum value     = 598
minimum value     = 0
// in this case 598 records were extracted
```

Mode 2 – Polygon Fill with Region Add

This mode is useful when trying to determine overlapping regions. As each polygon is being filled in, the value of the pixels inside the polygon gets increment by the fill value for that record. In this way if you assign the same fill value to every record, areas of overlap will have a greater value than the fill value.

Example: The following example illustrates the function.

```
z = ones(256,256)*100
M_plotcolor = 1
c = shp2image("d:\\temp\\sed.shp", z, -78.015::33.001::-
74.992::36.009, 0, 2)
view c
show c
--- c ---
data type is      : real
number of rows    = 256
number of columns = 256
maximum value     = 6
minimum value     = 0
// the maximun value of 6 indicates there were overlapping
regions;
// otherwise the maximum value would have been 1
```

shp2fillimage Shapefile Image Fill

Syntax: **shp2fillimage**($filename, image, bbox, recn)

Description: This function is useful when performing polygon fill of shapefile records of type 5 (polygons).

'$filename' is a string containing the name of the shapefile file,

'image' is an array variable used as input to the polygon fill routine (it serves as a background image),

'bbox' is a 1x4 array containing the bounding box of the region of interest (Xmin, Ymin, Xmax, Ymax).

bbox.text – optional input containing the 'proj4' string that describes the coordinate system of the vector features. Note: This option will be executed as long as there is a valid '.prj' file associated with the shape file. The '.prj' file can contain a WKT (well-known-text) representation of the coordiate system.

'recn' can be a Nx2 array, a Nx1 array or 0.

- If 'recn' is a Nx2 array, for each row, the first column contains the shapefile record id and the second column contains the value to use to fill in the corresponding polygon;
- if it is a Nx1 array, the column vector contains the shapefile ids to extract from the shapefile; and
- if it is 0, then all the shapefile records are extracted.

  The value used to fill in the polygons in the second and third case is determined by the content of the system variable M_plotcolor. The returned image is a combination of the input image and the filled in polygons.

Example: The following example illustrates the function.

```
z = ones(180,360)*100
$file = "../../react_data/global/place_find/admin98.shp"
c = shp2fillimage($file, z, -180::-90::180::90, 0)
view c
```

shp2vector Retrieve verticies from a shapefile

Syntax: **shp2vector**($dbf_filename, ID)

Description: Given a shapefile name (*.shp), return the verticies for the given shapefile object identifier, ID.

shpGetInfo provides information on a shapefile

Syntax: info = getshpinfo($filename)
or

info=shpGetInfo($filename)

Description: This function returns an array of information about the shapefile, the description of the output follows
file_cd= 9994,fileSize,version,shapeType,xmin,ymin,xmax,ymax.
If the shape file has an associated '.prj' file, then **info.text** will contain any coordinate system information, e.g. the Well-Known-Text or WKT string.

Example:

```
[ready]: getshpinfo("c:\\temp\\file.shp")
[ready]:
```

shpwriteroi Copy a subset of a shapefile

Syntax: **shpwriteroi**($shpfile, $dstshpfile,bbox)

Description: Copies the specified bounding box from $shpfile to a newly created shape file, $dstshpfile. Where

bbox = xmin:: ymin :: xmax::ymax

shiftt Shift an Array or Image

Syntax: **shiftt**(a,row,col)

Description: This function is used to shift or translate an input array or image by a specified number of columns and rows. The shift is non-cyclic, i.e., border pixels will not wrap-around. Note that the values of 'row'' and 'col' must be non-negative.

Example: The example on the next page generates an array and then applies the **shiftc** and **shiftt** functions.

```
[ready]: x = int(randu(3,3)*10)   // generate a 3 x 3 array
of random
[ready]: x                            // integers between 1
and 10
 (10^0) X
 row 0 =
  3.00   1.00   2.00
 row 1 =
  9.00   3.00   1.00
 row 2 =
  2.00   5.00   8.00
[ready]: shiftc(x,1,0)                // cyclic shift one row
down
 (10^0) X
 row 0 =
  2.00   5.00   8.00
 row 1 =
  3.00   1.00   2.00
 row 2 =
  9.00   3.00   1.00
[ready]: shiftt(x,0,1)                // translation one
column right
 (10^0) X
 row 0 =
  0.00   3.00   1.00
 row 1 =
  0.00   9.00   3.00
 row 2 =
  0.00   2.00   5.00
```

show *Display Variables Information*

Syntax: **show** variable [or list of variables]

Description: Display basic parameters of a list of arrays, or of all arrays in memory. Also have **show** all. Note that this command can not be used within an expression.

Example: Generate some variables and then invoke **show**,

```
[ready]: ramp = (0,255,1,256)
[ready]: a = randg(3,100)
[ready]: name = "Carlos"
[ready]: x = 5.00
[ready]: show all
    a            :  3   100 real    defined at level = 0
    x            :  1   1   real    defined at level = 0
    ramp         :  256 256 real    defined at level = 0
    name         :  1    6           string
[ready]: show a
       --- a ---
data type is      : real
number of rows    = 3
number of columns = 100
maximum value     = 2.75168
minimum value     = -2.92469
```

sign Sign of Array Elements

Syntax: **sign**(a) or a.**sign**

Description: Compute the sign of each element in the array. For every element in 'a' it returns **+1** if the element is greater than or equal to zero, and **-1** otherwise. This function does not accept complex inputs.

Example: **G**enerate a vector and then apply the **sign** function to the vector.

```
[ready]: a = geo(-2,7)           // generate a row vector
[ready]: sign(a)              // example of sign
row 0 =
  1.00  -1.00   1.00  -1.00   1.00  -1.00   1.00
```

sin Sine

Syntax: **sin**(a) or a.**sin**

Description: Compute the sine of each array element.
Note that the input is expected in radians.

sinc Sinc Function

Syntax: **sinc**(a) or a.**sinc**

Description: Evaluates the **sinc** function for each element in the input array. The **sinc** function is defined as

.

Note that sinc(0) is evaluated as 1.

---

sinh Hyperbolic Sine

Syntax: **sinh**(a) or a.**sinh**

Description: Computes the hyperbolic sine of each array element.

The hyperbolic sine of x is evaluated as

$$\sinh(x) = \frac{e^{x} - e^{-x}}{2}.$$

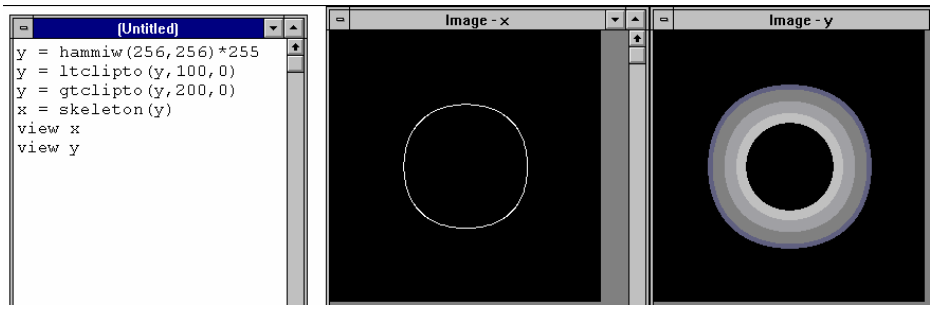
---

skeleton Binary Conversion Filter

Syntax: **skeleton**(array)

Description: Computes the skeleton on an input image.

Example:



smodify String Replace

Syntax: **smodify(**$string**,**$from**,**$to**)**

Description: This replaces all occurrences of $from with $to in the string $string.

The output of this function can be set to a resultant string also.

Example: This replaces the 'F' in 'Fun' with a 'G' to form "Gun'.

```
[ready]: smodify("Fun","F","G")          // replaces "F"
with "G"
Gun
```

sortr Row Wise Sorting

Syntax: **sort**r(a) or a**.sortr**

Description: Returns a sorted version, on a row basis, of the input array. The sorting algorithm utilized is 'Heapsort'. The sorting is performed from small number to large number.

Example: Create an array and then perform a row wise sort of the array.

```
[ready]: a=randu(2,7)        // create random 2x7 matrix
[ready]: a
row 0 =
  0.41   0.72   0.63   0.39   0.62   0.50   0.98
row 1 =
  0.79   0.17   0.01   0.20   0.42   0.10   0.28
[ready]: sortr(a)            // example of sort
row 0 =
  0.39   0.41   0.50   0.62   0.63   0.72   0.98
row 1 =
  0.01   0.10   0.17   0.20   0.28   0.42   0.79
```

spatf Spatial Filter

Syntax: **spatf**(a,n,m,function)

Description: Applies a user selected (or user defined) spatial filter to the input array . The filter is implemented over a moving window through out the whole input array. The dimensions of the window are (N x M). The window must be able to fit within the dimensions of 'a', otherwise an error message will be generated. The final argument, 'function', is the name of a single argument internal MSHELL function which returns a scalar value, e.g., var, mean, max, min, ...

Example: This module provides significant flexibility for performing arbitrary spatial filtering functions. For example, the local mean and variance over the array 'a', using a N x N window, can be computed, respectively, using the following MSHELL calls,

```
spatf(a,N,N,mean)
spatf(a,N,N,var).
```

Likewise, the local maximum over a 9 x 9 window can be computed with

```
spatf(a,9,9,max).
```

sqlite_create Creates an empty SQLite database

Syntax: sqlite_create($db_filename)

Description: This function creates an empty SQLite database, and stores it in $db_filename (absolute path). Note that $db_filename is created by the sqlite_create() function and must not exist before.

If successful, it returns 0, otherwise it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

sqlite_connect Connects to an SQLite database

Syntax: **sqlite_connect**($database)

Description: Use this command to connect to an SQLite database. The string passed is the file name of an existing SQLite database. If successful, it returns 0, otherwise it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

Example: see sqlite_getrows example.

sqlite_close Closes access to an SQLite database

Syntax: **sqlite_close**()

Description: Use this command to close the already opened SQLite database. If successful, it returns 0, otherwise it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

Example: see sqlite_getrows example

sqlite_exec Sends a query to the SQLite database

Syntax: **sqlite_exec**($query)

Description: This function executes a query on the opened SQLite database and it is used after an sqlite_connect . If successful, it returns 0, otherwise it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

Example: see sqlite_getrows example.

sqlite_getcolnames Returns column names in a result set

Syntax: **sqlite_getcolnames**()

Description: Returns a comma separated list of the column names in a result set. To be called after a SELECT query call to sqlite_exec(). Any expressions in the SELECT statement may be returned as an empty column name, unless the AS operator is used. The column name should match the AS identifier properly.

Example: see sqlite_getrows example.

sqlite_getrows Returns the result set as a string data

Syntax: **sqlite_getrows(**srow,erow**)**

sqlite_getrows (srow,erow,$fname**)**

sqlite_getrows ($colnames,srow,erow**)**

sqlite_getrows ($colnames,srow,erow,$fname**)**

where,

srow – start row number

erow – end row number

$fname – output file name (string)

$colnames – list of requested field names

Description: This function works after a sqlite_connect and sqlite_exec. After a query this function when invoked will return the result set data as a string. The columns are separated by a | and rows are separated by new lines. This function may be called multiple times on the same result set. There are different ways to extract the results of the query using this function. If an error occurs, it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

Example: The following example shows most of the SQLite functions being utilized on a simple test case.

```
status = sqlite_connect("test.sqlite")
if (status) {
          status.text
          return
}
$query = "select * from RobinsonTargets"
status = sqlite_exec($query)
if (status) {
          status.text
          return
}
nrec = sqlite_rowcount()

srow = 1
erow = 3
$cols = "site_name,lat1,lon1"
$filename = "sqlite.txt"

s1hdr = sqlite_getrows(0, 0)
if (s1hdr) {
          s1hdr.text
          return
}
s1 = sqlite_getrows(srow, erow)
if (s1) {
          s1.text
          return
}
print "SQLITE Test 1: extraction of records
"::int2str(srow)::" through "::int2str(erow):: ".\n"
print s1hdr.text::s1.text::"\n"

s2hdr = sqlite_getrows($cols, 0, 0)
if (s2hdr) {
          s2hdr.text
          return
}
s2 = sqlite_getrows($cols, srow, erow)
if (s2) {
          s2.text
          return
}
print "SQLITE Test 2: extraction of records
"::int2str(srow)::" through "::int2str(erow):: " following
provided column names.\n"
print s2hdr.text::s2.text::"\n"

s3hdr = sqlite_getrows(0, 0, $filename)
if (s3hdr) {
          s3hdr.text
          return
}
```

```
s3 = sqlite_getrows(srow, erow, $filename)
if (s3) {
          s3.text
          return
}
print "SQLITE Test 3: extraction of records
"::int2str(srow)::" through "::int2str(erow):: ". Result is
also written into the specified file.\n"
print s3hdr.text::s3.text::"\n"

s4hdr = sqlite_getrows($cols, 0, 0, $filename)
if (s4hdr) {
          s4hdr.text
          return
}
s4 = sqlite_getrows($cols, srow, erow, $filename)
if (s4) {
          s4.text
          return
}
print "SQLITE Test 4: extraction of records
"::int2str(srow)::" through "::int2str(erow):: " following
provided column names. Result is also written into the
specified file.\n"
print s4hdr.text::s4.text::"\n"

$cols = "lat1,lon1"
erow = 10
s5 = sqlite_getrowsfloat($cols, srow, erow)
print "SQLITE Test 5: extraction of records
"::int2str(srow)::" through "::int2str(erow):: " as an ARRAY
variable.\n"
s5

status = sqlite_close()
```

sqlite_getrowsfloat *Returns the result set as an array*

Syntax: **sqlite_getrows(**srow,erow**)**

sqlite_getrows (srow,erow,$fname**)**

sqlite_getrows ($colnames,srow,erow**)**

sqlite_getrows ($colnames,srow,erow,$fname**)**

where,

srow – start row number

erow – end row number

$fname – output file name (string)

$colnames – list of requested field names

Description: This function works after a sqlite_connect and sqlite_exec. After a query this function when invoked will return the result set data as an array. The first input argument $column_name is a comma separated list of column names to be retrieved. This function may be called multiple times on the same result set. If an error occurs, it returns a NULL array and the text attribute can be used to retrieve a brief description of the error.

Example: see sqlite_getrows example

---

sqlite_rowcount Returns the result set count

Syntax: **sqlite_rowcount**()

Description: This function works after an sqlite_connect and sqlite_exec. After a query this function when invoked will return the number of rows in the result set.

Example: see sqlite_getrows example

---

sqlite_transact Transacts with an SQLite database

Syntax: **sqlite_transact**($query)

Description: Use this command to post the passed string as the next query for the previously opened SQLite database. The output of the query, if any, is automatically retrieved. The fields are delimited by the character '|'. The output is assigned to a string variable. If successful, it returns 0 and the text attribute holds the query result string. If an error occurs, it returns a non-zero value and the text attribute can be used to retrieve a brief description of the error.

Example:

```
status    = sqlite_connect("test.sqlite")
$query    = "select site_name, lat1, lon1 "
$query  = $query::"from RobinsonTargets"

$result = sqlite_transact($query)
$result

status    = sqlite_close()
```

sqrt Square Root

Syntax: **sqrt**(a) or a.**sqrt**

Description: Computes the square root of each array element in 'a'. If any of the entries in the input array are negative the output of the square root will be complex, i.e. sqrt(-1) = 0+1i, where i implies that the number is imaginary. Taking the square root of a complex array is a valid operation.

stats Computes Array Basic Statistics

Syntax: **stats**(a) or a.**stats**

Description: Returns a row vector whose first element is the minimum value in 'a', its second element is the maximum value in 'a', its third element is the mean value of the elements of 'a', and its last element is the standard deviation of the elements of 'a'.

statxs Computes Statistics with exclusion vlaues

Syntax: **statsx**(a,exclude)

Description: Returns a row vector whose first element is the minimum value in 'a', its second element is the maximum value in 'a', its third element is the mean value of the elements of 'a', and its last element is the standard deviation of the elements of 'a'. The statistics are computed omitting any values present in the 'exclude' array.

stop_process terminates execution with message

Syntax: **stop_process**($error_message)

Description: forces a script to stop its execution (similar to hitting the ESC key) and uses the user provided $error_message string as the message that will initialize the system variable

M_errorstring. This function is useful when errors are being trapped by M_errorscript, since the user provided $error_message can be recorded in a database or in a log file. .

STDOUT Standard out identifier

Syntax: **writecolor**(**STDOUT**, r, g, b, "TIFF")

Description: Reserved word used to specify the "Standard Out" instead of a file name. (Only works for the console and web versions of MSHELL, not the GUI)

Example: The following MSHELL code will write a JPEG image for a web request coming from an <IMG SRC=…> tag.

```
print "Content-type:  image/jpeg\n\n"
writecolor(STDOUT, red_plane, green_plane, blue_plane,
"JPEG")
```

str2float Converts Numeric String to Float

Syntax: **str2float**($string)

Description: Returns an array where numeric values in the string, delimeted by commas or linefeeds, are converted to float values and then saved in an array variable. Values delimeted by commas are stored in the same array row. The linefeed , '\n' , is used as the row delimeter when copying into the array.

See Also: **str2int**, **strlen , str2float, strhex2int, float2str**

Example: The following is an example of **str2float** usage with a 2x3 array creation.

```
ready]: $str = "4.56,34.25,67.02\n12.28,34.64,3.23"          //define string variable
[ready]: str2float($str)         // example of str2float

(10^0) X
row 0 =
  4.56  34.25  67.02
row 1 =
 12.28  34.64   3.23
```

str2int Converts Numeric String to Integer

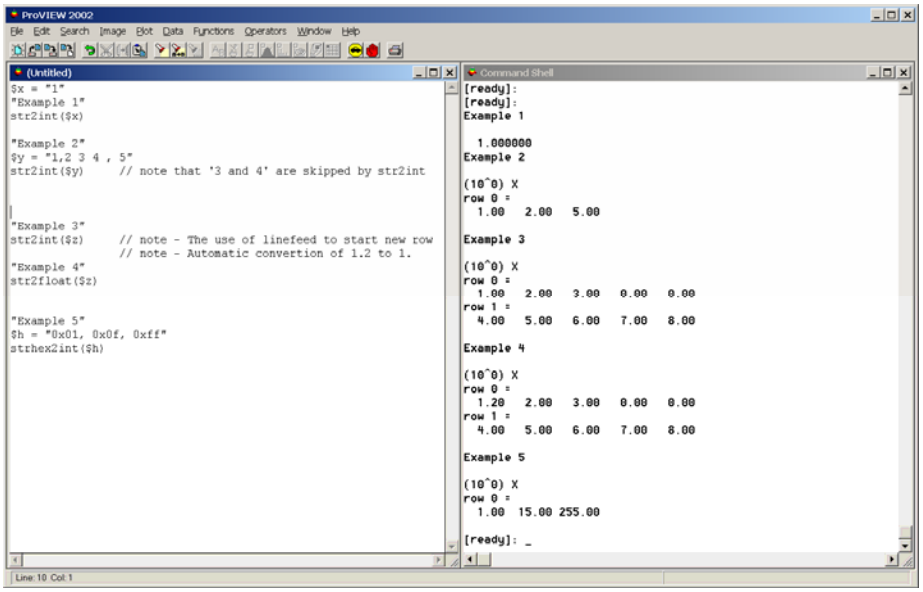
Syntax: **str2int**($string)

Description: Returns an array where numeric values in the string, delimeted by commas or linefeeds, are converted to integer values and then saved in an array variable. Values delimeted by commas are stored in the same array row. The linefeed , '\n' , is used as the row delimeter when copying into the array.

See Also: **str2float**, **strlen, str2float, strhex2int, int2str**

Example: See code below for example usage of str2int(), st2float(), and strhex2int().

```
"Example 1"
str2int($x)

"Example 2"
$y = "1,2 3 4 , 5"
str2int($y)   // note that '3 and 4' are skipped by str2int



"Example 3"
str2int($z)   // note - The use of linefeed to start new row
              // note - Automatic convertion of 1.2 to 1.
"Example 4"
str2float($z)


"Example 5"
$h = "0x01, 0x0f, 0xff"
strhex2int($h)
```



strContains check for substring occurance

Syntax: **strContains(**$a,$pattern**)**

Description: This function return the number of times substring '$pattern' occurs in the input string '$a'.

Example: see strSection( ) example.

strEndsWith check for string termination match

Syntax: **strEndsWith(**$a,$pattern**)**

Description: This function return the value of 1, if it finds the string pattern '$pattern', at the end of the input string '$a'. Otherwise it returns the value 0.

Example: see strSection( ) example.

strlen Computes the Length of a String

Syntax: **strlen**($string)

Description: Returns a 1 x 1 array with the number of characters in a $string.

See Also: **str2float**, **str2ind**

Example: The following generates a string variable and then calculates **str2float**, **sstr2ind**, and **strlen** of the variable.

```
[ready]: $strvalue = "4.356"          // define string
variable
[ready]: str2float($strvalue)         // example of str2float
  4.356
[ready]: str2int($strvalue)           // example of str2int
  4
[ready]: strlen($strvalue)                // example of
srtlen
  5
```

strhex2int Converts Hex String to Integer

Syntax: **strhex2int**($string)

Description: Returns an array where hex values in the string, delimeted by commas or linefeeds, are converted to integer values and then saved in an array variable. Values delimeted by commas are stored in the same array row. The linefeed , '\n' , is used as the row delimeter when copying into the array.

See Also: **str2float**, **strlen, str2int**

Example: See str2int() example.

strlow2up Converts lowercase to uppercase

Syntax: **strlow2up(**$lowercase**)**

Description: The output is a string whereby all lowercase characters have been changed to uppercase.

strStripWhiteSpace Removes whitespaces at the beginning and at the end of a string

Syntax: strStripWhiteSpace($to_be_stripped)

Description: This function removes the whitespaces characters (along with \t, \n and everything that matches (\s) in a regular expression) at the beginning and at the end of the input string.

The result is returned as a string.

strStartsWith check for string start match

Syntax: **strStartWith(**$a,$pattern**)**

Description: This function return the value of 1, if it finds the string pattern '$pattern', at the start of the input string '$a'. Otherwise it returns the value 0.

Example: see strSection( ) example.

strSection extract section of a String

Syntax: **strSection(**$a,$delimiter,start,end**)**

Description: This function mimics Qt's QString::section() function. Given an input string '$a', and a user defined string delimiter '$delimiter', it allows the user to treact the input string as sections, and extract all sections from 'start' to 'end'. This function is quite usefull to break apart commad-space-value (CSV) type of input strings. From left to right section are identified as 0,1,….. From right to left section are identified as -1,-2,….

Example:

```
$a = "my life as a dog"
strStartsWith($a,"my")        // returns 1
strStartsWith($a,"mi")        // returns 0

strEndsWith($a,"as a dog")    // returns 1
strEndsWith($a,"as a cat")    // returns 0

strContains($a,"a")       // returns 2
strContains($a,"lice")        // returns 0

$a = "my life , as a , dog"
strSection($a,",",1,1)
```

strup2low Converts uppercase to lowercase

Syntax: **strlow2up(**$uppercase**)**

Description: The output is a string whereby all uppercase characters have been changed to lowercase.

sum *Sum All Elements*

Syntax: **sum**(a) or a.**sum**

Description: Sum all the elements in the input array. The output of this operation is a scalar, i.e. a 1 x 1 array.

Example: With example for **sumr**.

sumc *Sum Column Vectors in an Array*

Syntax: **sumc**(a) or a.**sum**c

Description: For each row in the input array, sum the values along the row. The output is a column vector.

Example: With example for **sumr**.

sumcum *Row Wise Cumulative Sum*

Syntax: **sumcum**(a) or a.**sumcm**

Description: For each row in the input array, compute the cumulative sum of values along the row. The output will have the same dimensions as the input.

Example: With example for **sumr.**

sumr *Sum Row Vectors in an Array*

Syntax: **sum**r(a) or a.**sum**r

Description: For each column in the input array, sum the values along the column. The output is a row vector.

Example: The following generates an array and then calculates **sum**, **sumc**, **sumcum**, and **sumr** of that array.

```
[ready]: x = randu(3,3)*10        // generate a 3 x 3 array
of random
[ready]: x = nint(x)          // integers in the range of 1
to 10
row 0 =
  2.00   4.00   2.00
row 1 =
  1.00   7.00  10.00
row 2 =
  5.00   5.00   5.00
[ready]: sum(x)               // example of sum(x)
 41.00
[ready]:
[ready]: sumc(x)              // example of sumc(x)
row 0 =
  8.00
row 1 =
 18.00
row 2 =
 15.00
[ready]:
[ready]: sumcum(x)                // example of sumcum(x)
row 0 =
  2.00   6.00   8.00
row 1 =
  1.00   8.00  18.00
row 2 =
  5.00  10.00  15.00
[ready]: sumr(x)              // example of sumr(x)
row 0 =
  8.00  16.00  17.00 [ready]:
```

svd Singular Value Decomposition

Syntax: **svd**(A,U,D,V)

Description: Computes the singular value decomposition of an input matrix. The routine used for this computation is derived from "Numerical Recipes in C", see page 3, "**References and Further Readings**". This routine takes the input matrix, 'A', and returns 'U', 'D', and 'V', which contain the singular value decomposition of 'A'. The following equalities will hold:

```
U'*U = 1
V'*V = 1
A = U*D*V'
```

Note that all the input variables must exist before calling this function.

Example: Performs the operation and then checks the result.

```
[ready]: a = randu(2,2)   // create a 2 x 2 random array
[ready]: a
row 0 =
  0.23   0.36
row 1 =
  0.16   0.55
[ready]: u=0; d=0; v=0        // initialize u,d,v prior to
svd call
[ready]: svd(a,u,d,v)     // call svd function
[ready]: u*d*v'           // test output
row 0 =
  0.23   0.36
row 1 =
  0.16   0.55
```

system Issues Operating System Command

Syntax: **system** string

Description: This function allows you to issue an operating system command. It invokes the operating system command processor to execute a command, batch file, or other program named by the string command. To be located and executed, the program must be in the current directory or in one of the directories listed in the PATH string in the environment. In the case of the string consisting of a an executable command, batch file, or program, you will be returned to the MSHELL prompt upon completion of the executable. If the the string consists of a non-executable statement or is null then returning to MSHELL will require you to type " exit" at the DOS prompt. The COMSPEC environment variable is used to find the command processor program file, so that file need not be in the current directory.

Example: The following operations illustrate the function.

```
[ready]: system "edit c:/autoexec.bat"    // go to DOS, edit
the file
[ready]:                                  // when done
returns to the [ready]:                                  //
ProVIEW prompt directly.
[ready]:
[ready]: system                           // opens a DOS
window:
c:\>rem       at the DOS prompt you can execute operating
c:\>rem       system commands, batch files, or other DOS
c:\>rem       programs; when done type EXIT
c:\>exit  rem     to return to the ProVIEW prompt
[ready]:                                  // ready to
continue working
[ready]:                                  // in ProVIEW
[ready]:
[ready]: system "dir/s/b > dir.lst"       // creats and
stores a listing
[ready]: $list = readtext("dir.lst")      // of files in a
directory
```

sys_ Issues Operating System Commands

In this group you will find the following system functions:

sys_pwd(…), sys_cd(…), sys_mkdir(…), sys_dir(…), sys_move(…), sys_ren(…), sys_copy(…), sys_del(…), sys_deltree(…), sys_dirsize(…), sys_filesize(…), sys_sendmail(…)

Detail syntax and examples are provided below.

sys_pwd Prints the current working directory

Syntax: **sys_pwd**()

Description: This function returns a string containing the absolute path of the current working directory.

Example: Printing the current directory.

```
$dir = sys_pwd()

print "Current directory is "::$dir
```

sys_sendmail Sends mail

Syntax: **sys_sendmail**($srvrIP,$from,$to,$subject,$message)

Description: This function sends an email to email address specified in '$to'. The user must provide the IP address of the mail server in the variable $srvrIP. **This function does not resolve the name of the machine, it must be the IP address.** The function will return a status value. The text component of the status can be checked for any error message. The input parameters are:

$srvrIP - IP address of mail server to use

$from - email address of sender

$to - email address of receiver

$subject - subject string

$message - message string

Example: Sending email

```
$srvrIP                = "207.87.28.34"
$from                  = "me@actgate.com"
$to                    = "info@actgate.com"
$subject               = "test"
$message               = "just a test of sendmail"

status = sys_sendmail($srvrIP, $from, $to, $subject, $message);
status.text
```

sys_cd Changes the current directory

Syntax: **sys_cd**($newdir)

Description: This function changes the current working directory to $newdir. If successful, it returns 0, otherwise it returns a non-zero value and the text attribute is set with a brief description of the error (see the table below).

Return value	***".text" attribute***	***Meaning***
0	Not set	Success
1	Set	The directory does not exist
2	Set	The argument is not a directory
3	Set	Cannot access the directory

Example: Changing the current directory.

```
$dir = sys_pwd()
print "Current directory (before) "::$dir

s = sys_cd(“temp”)

if (s) {
          print “An error occurred... ”
          s.text
          return
}

$dir = sys_pwd()
print "Current directory (after) "::$dir
```

sys_mkdir Creates a directory

Syntax: **sys_mkdir**($newdir)

Description: This function create the directory $newdir. If successful, it returns 0, otherwise it returns a non-zero value and the text attribute is set with a brief description of the error (see the table below).

Return value	***".text" attribute***	***Meaning***
0	Not set	Success
1	Set	File already exists
2	Set	Directory already exists
3	Set	Object already exists
4	Set	Permission denied

Example: Creating a new directory.

```
s = sys_mkdir("data")

if (s) {
          print "An error occurred ... "
          s.text
          return
}
```

sys_dir Directory listing

Syntax: **sys_dir**($thedir)

Description: This function returns a string containing the list of files and directories located within the directory $thedir; if $thedir is an empty string, the function lists the content of the current directory. Also if $thedir does not exists, is not a directory or is not readable, NULL is returned.

Example: Showing the content of a directory.

```
// listing the current directory
sys_dir("")

// listing the content of "data"
$list = sys_dir("data")

if ($list == "") {
          print "An error occurred... "
          return
}

print $list
```

sys_dirlist Controlled directory listing

Syntax: **sys_dirlist($path,$options)**

Description: This function returns the list of the subdirectories or files of a given directory. The first input argument is a string, containing the directory name. The second input argument is one of the following "d" , "s" or, "p" , where:

- "d" - if used the function will return only the list of subdirectories. Otherwise only the files are listed.
- "s" – if used the function will return the list of files along with their size.
- "p" – if used the function will list the path of the files .

Note: this function uses '|' as the delimiter for each field in a record.

Example:

```
// to print directories, their sizes, and the full path
$out = dirlist("c:\\temp\\*.*", "dsp" )

// to print the directory listing only.  This will not print
files
$x = dirlist("F:/crism_pds_archive/trdr/TRDR/*","d")
```

sys_move Moves or renames files or directories

Syntax: **sys_move**($source, $dest)

Description: If the $dest argument is a directory, this function moves the $source argument in it. Otherwise $source is renamed to $dest. Note that $source can be either a file or a directory. If successful, it returns 0, otherwise it returns a non-zero value and the text attribute is set with a brief description of the error (see the table below).

Return value	***".text" attribute***	***Meaning***
0	Not set	Success
1	Set	$source does not exist
2	Set	Permission denied (case: $source --> $dest)
3	Set	Permission denied (case: $source --> $dest/$source)
4	Set	Cannot move to a non directory
5	Set	First argument is neither file nor directory

Example: Moving and renaming files and/or directories.

```
// rename file or directory
s = sys_move(“file1.txt”, “better_name.txt”)
s = sys_move(“dir1”, “newdir”)

// moves “file2.dat” to “data_dir/file2.dat”
s = sys_move(“file2.dat”, “data_dir”)

// moves “old_tree” to “backup/old_tree”
s = sys_move(“old_tree”, “backup”)


// to catch errors:
if (s) {
          print “An error occurred... ”
          s.text
          return
}
```

sys_ren Renames files or directories

Syntax: **sys_ren**($oldname, $newname)

Description: This function just renames the $oldname (it can be either a file or a directory) to $newname. Note that this function is provided for convenience, but the same functionality can be achieved with "sys_move". If successful, it returns 0, otherwise it returns a non-zero value and the text attribute is set with a brief description of the error (see the table below).

Return value	***".text" attribute***	***Meaning***
0	Not set	Success
1	Set	First argument does not exist
2	Set	Permission denied
3	Set	Second argument already exists
4	Set	First argument is neither file nor directory

Example: Renaming files and/or directories.

```
// rename files or directories
s = sys_ren("file1.txt", "better_name.txt")
s = sys_ren("dir1", "newdir")


// to catch errors:
if (s) {
          print "An error occurred... "
          s.text
          return
}
```

sys_copy *Copies files*

Syntax: **sys_copy**($source, $dest)

Description: If the $dest parameter is a directory, this function copies the file $source in it. Otherwise the file $source is copied to a new file named $dest (in the current directory). If successful, it returns 0, otherwise it returns a non-zero value and the text attribute is set with a brief description of the error (see the table below).

Note that $dest is meant to be relative to the current working directory (see the sys_pwd function). Absolute paths should not be used in $dest.

Return value	***".text" attribute***	***Meaning***
0	Not set	Success
1	Set	Source file does not exists
2	Set	First argument is not a file
3	Set	Permission denied (cannot create $dest)
4	Set	Permission denied (cannot create $dest/$source)
5	Set	Last argument exists but it is not a directory

Example: Copying files.

```
// makes a backup copy
s = sys_copy(“results.txt”, “results_copy.txt”)

if (s) {
          print “An error occurred... ”
          s.text
          return
}

// copies “file.dat” to “destdir/file.dat”
s = sys_move(“file.dat”, “destdir”)

if (s) {
          print “An error occurred... ”
          s.text
          return
}
```

sys_del *Deletes files or empty directories*

Syntax: **sys_del**($filename)

Description: This function attempts to remove $filename ($filename can be either a file or an empty directory). If successful, it returns 0, otherwise it returns a non-zero value and the text attribute is set with a brief description of the error (see the table below).

Return value	***".text" attribute***	***Meaning***
0	Not set	Success
1	Set	$filename does not exists
2	Set	File access denied
3	Set	Directory not empty
4	Set	Directory access denied
5	Set	Cannot delete the object

Example: Deleting files and (empty) directories.

```
// file
s = sys_del("data.txt")

// directory
s = sys_del("mydir")

if (s) {
          print "An error occurred... "
          s.text
          return
}
```

sys_deltree Deletes an entire directory tree

Syntax: **sys_deltree**($dirname)

Description: This function attempts to recursively remove the directory $dirname along with all its content. If successful, it returns 0, otherwise it returns a non-zero value and the text attribute is set with a brief description of the error (see the table below).

NOTE: If an error (e.g. “Permission Denied”) occurs while deleting a file or a sub-directory, the function returns with error 4, but files and directories removed before the error has occurred are permanently deleted.

Return value	***“.text” attribute***	***Meaning***
0	Not set	Success
1	Set	$dirname does not exists
2	Set	$dirname is not a directory
3	Set	Cannot remove current or parent directory
4	Set	Cannot delete the entire tree
5	Set	Directory access denied

Example: Deleting a directory tree.

```
s = sys_deltree(“old_data”)

if (s) {
          print “An error occurred... ”
          s.text
          return
}
```

sys_dirsize Returns the size of a directory tree

Syntax: **sys_dirsize**($dirname)

Description: Returns the size of the entire $dirname directory tree. The total size is computed by summing the sizes of all files recursively found within $dirname. If an error occurs, it returns a negative value and the text attribute is set with a brief description of the error (see the table below).

Return value	***".text" attribute***	***Meaning***
Size in bytes	Not set	Success
-1	Set	$dirname does not exists
-2	Set	$dirname is not a directory
-3	Set	Error reading the sub-tree
-4	Set	Directory access denied

Example: Reading the file size of a directory tree.

```
// size of the current directory
size = sys_dirsize(“.”)

// size of the “data” directory
size = sys_dirsize(“data”)

if (size < 0) {
          print “An error occurred... ”
          size.text
          return
}
```

sys_filesize Returns the size of a file

Syntax: **sys_filesize**($filename)

Description: Returns the size of an existing file in bytes. If an error occurred, it returns a negative value and the text attribute is set with a brief description of the error (see the table below).

Return value	***".text" attribute***	***Meaning***
Size in bytes	Not set	Success
-1	Set	$filename does not exists
-2	Set	$filename is not a file
-3	Set	$filename is not readable

Example: Reading the file size.

```
$file = "fatfile.dat"
size = sys_filesize($file)

if (size < 0) {
          print "An error occurred... "
          size.text
          return
}

print $file::" size is "::int2str(size)
```

sys_cat Concatenates input files

Syntax: **sys_cat**($input_files, $output_file)

Description: Concatenates the input files and writes the result in the specified output file. The input files list ($input_files) is comma separated. If an error occurs, it returns an exit code (greater than zero) with the text attribute containing a brief description of the error (see the table below).

Return value	***".text" attribute***	***Meaning***
0	Not set	Success
1	Set	$input_files: some do not exist
2	Set	$input_files: is not a list of files
7	Set	Cannot open $output_file
6	Set	Cannot read some $input_files
5	Set	$output_file already exists

Example:

```
$inputfiles = "fileA.txt,fileB.txt"
$out = "merged.txt"
status = sys_cat($inputfiles, $out)

if ( status ) {
          print "An error occurred... "
          status.text
          return
}
```

- T -

tan *Tangent*

Syntax: **tan**(a) or a.**tan**

Description: Compute the tangent of each array element. Note that the input is expected in radians.

tanh *Hyperbolic Tangent*

Syntax: **tanh**(a) or a.**tanh**

Description: Compute the hyperbolic tangent of each array element.

The hyperbolic tangent of 'a' is evaluated as

$\tanh(a_{j,i}) = \sinh(a_{j,i}) / \cosh(a_{j,i})$ for all i, j;

where j and i are, respectively, row and column idices.

textoverlay *Annotates an Image with Text*

Syntax: **textoverlay**(image, $text, "Font", PointSize**::**row#**::** col#**::** color**::** overlay**::** orientation)

Description: This is used to place a text overlay on an image where $text is the string to be displayed, and "Font" is a string specifying the font to use. Row # and column# are the corresponding starting pixels for the text on the image. Color is the color of the text with the folowing possibilities:

0 = black

1 = dark gray

2 = blue

3 = cyan

4 = gray

5 = yellow

6 = magenta

7 = red

8 = green

9 = white

The actual image is not modified with use of this function. It is just

an overlay.

Overlay determines the background of the text where 0 gives a background of white or 1 gives a transparent background. Orientation specifies whether the text is vertical (0) or horizontal (1).

---

textremove Removes Text annoation

Syntax: **textremove**(image,$text)

Description: Used to remove the desired string ($text) from an image if it was created using the command **textoverlay**. Only works under MSHELL, i.e. the GUI mode.

---

text2image Converts Text to an Image

Syntax: **text2image**($sexpr)

Description: Given a single line input string, **text2image** converts the input string into a two dimensional array, where 'sexpr' is a string expression Once the text exists in image form it can be manipulated just as any other image.

Example: The ProVIEW screen of Figure 5 illustrates this application.

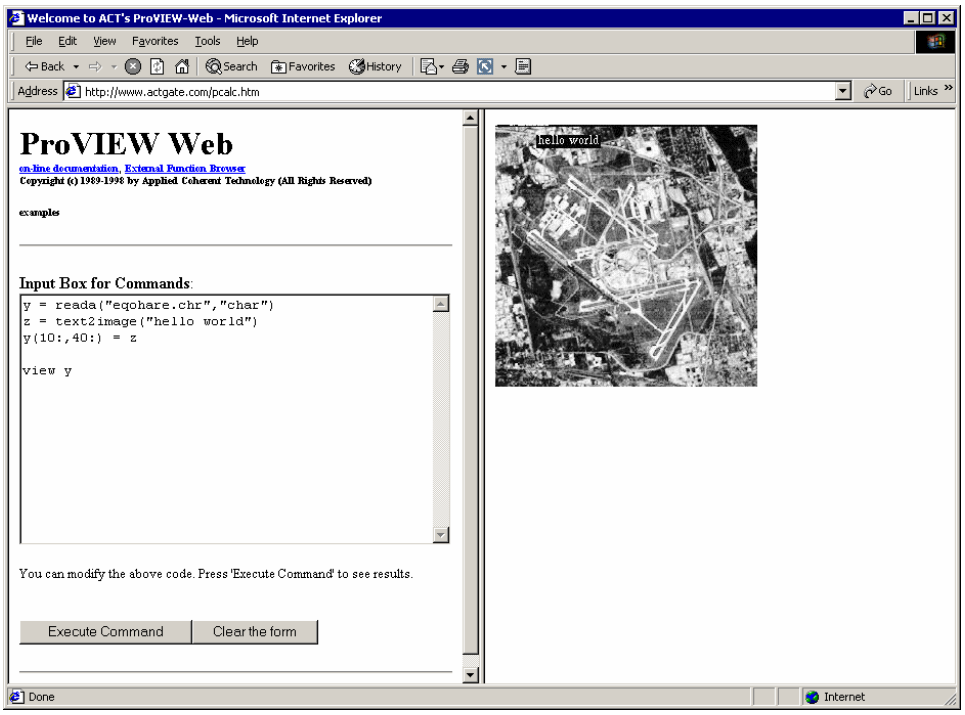


Figure 5

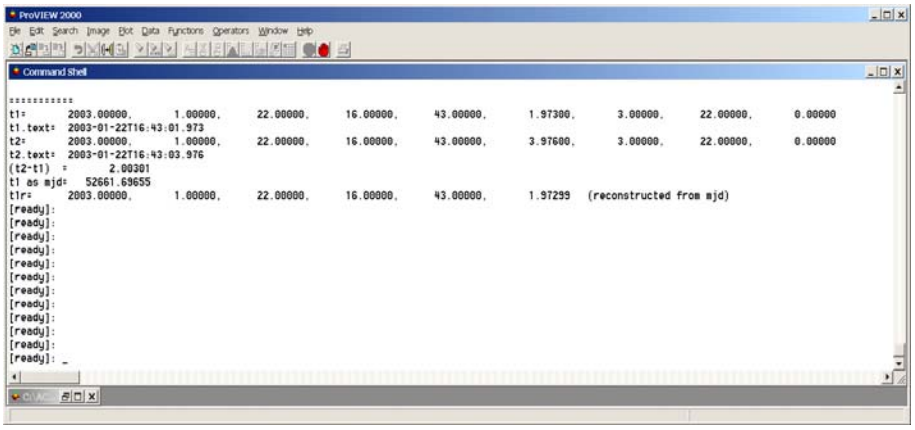
tdiff *computes elapse time*

Syntax: **tdiff(tend,tstart)**

Description: computes the elapse time, in seconds, between two times points.

Example: The following example illustrates the use of localtimenow(), tdiff(), and dt2mjd(). The function tdiff() makes all its internal computations in double precision.

```
M_format = "000000000.00000"

t1     = localtimenow()
pause(2)
t2     = localtimenow()

print "===========\n"
print "t1=\t"::float2str(t1)::"\nt1.text=\t"::t1.text::"\n"
print "t2=\t"::float2str(t2)::"\nt2.text=\t"::t2.text::"\n"
print "(t2-t1)  ="::float2str(tdiff(t1,t2))::"\n"

t1mjd=dt2mjd(t1)     // compute modified julian date for t1
print "t1 as mjd="::float2str(t1mjd)::"\n"
print "t1r=\t"::float2str(mjd2dt(t1mjd))::"(reconstructed)\n"
```



trace *Sum Diagonal Elements*

Syntax: **trace**(a) or a.**trace**

Description: Sums the elements on the diagonal of the input square array.

Example: Generate a square array and calculate **trace.**

```
[ready]: x = randg(3,3)
[ready]: x
row 0 =
    -0.85      0.94      0.16
row 1 =
     1.23     -1.08     -0.46
row 2 =
    -0.88     -0.80      0.41
[ready]: trace(x)
        -1.52219
```

- U -

url_copy_to_file Save a remote resource to a file

Syntax: status = **url_copy_to_file**($url, $local_file)

Description: This command can be used to retrieve a remote resource and save it to the specified local file path. It returns 0 upon success, 1 otherwise. In case of an error the text attribute of the returned ARRAY contains a brief description of the error.

Example:

```
[ready]: status =
url_copy_to_file("http://www.actgate.com/home",
"c:\\temp\\act-homepage.htm")
[ready]: status
    0.00
[ready]:
```

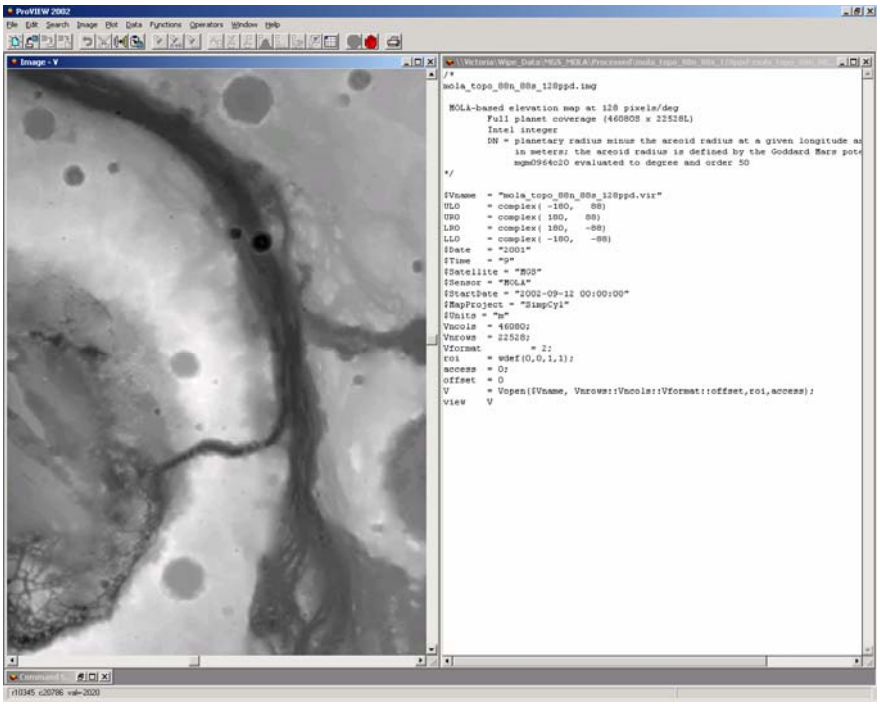
- V -

V Virtual Variable

Syntax: **V**

Description: This variable is used to hold the association to the virtual file. All access to any array opened virtually will be performed using this variable.

Example: The following screen shows the usage of the virtual format to read a large mosaic of Mars (ncols = 46080, nrows= 22528). Only a portion of the file is in the display window. As the user moves the scroll bar, it gets resposition to the right viewing location on the map.


ProVIEW 2002
Image - V
/*
mola_topo_88n_88s_128ppd.img
MOLA-based elevation map at 128 pixels/deg
Full planet coverage (46080S x 22528L)
Intel integer
DN = planetary radius minus the areoid radius at a given longitude a
in meters; the areoid radius is defined by the Goddard Mars pote
mgm0964c20 evaluated to degree and order 50
*/
$Vname = "mola_topo_88n_88s_128ppd.vir"
ULO = complex(-180, 88)
URO = complex(180, 88)
LRO = complex(180, -88)
LLO = complex(-180, -88)
$Date = "2001"
$Time = "9"
$Satellite = "MGS"
$Sensor = "MOLA"
$StartDate = "2002-09-12 00:00:00"
$MapProject = "SimpCyl"
$Units = "m"
Vncols = 46080;
Vnrows = 22528;
Vformat = 2;
roi = wdef(0,0,1,1);
access = 0;
offset = 0
V = Vopen($Vname, Vnrows::Vncols::Vformat::offset,roi,access);
view V

See Also: Vopen, Vnew, Vclose,

Vclose Closes Virtual Variable Link

Syntax: **Vclose**(V)

Description: This function is used to close the link between the virtual variable 'V' and the disk file it was previously attached to by **Vopen**.

See Also: **Vnew**,and **Vopen.**

Vnew Makes Disk Space for Virtual Image

Syntax: **Vnew**($fname, nrows,ncols,bytes_per_pixel)

Description: This function is used to allocate disk space in a disk file, where: '$fname' is the string variable holding the file name to use, 'nrows' is the number of rows in $fname, 'ncols' is the number of columns in $fname, and 'bytes_per_Pixel' is the number of bytes per pixel in '$fname'. Returns status of **0** for **good**.

See Also: **Vclose**, **Vopen**.

Vopen Makes Virtual Variable Link to File

Syntax: V = Vopen($fname,M::N::format::offset,roi,access);

Description: This function establishes a link between the Virtual Variable 'V' and a disk file '$fname', where: M is the number of rows in '$fname', 'N' is the number of columns in '$fname', byte 'format' is determined by the list below, 'offset' is the header size in bytes, and 'roi' is a valid rectangular region of interest within '$fname'. Also the access parameter has been added to allow just read (access = 0) or read and write (access = 1).

MSHELL Virtual variable formats:

Format 1: Byte

Format 2: PC16

Format 3: Sun16

Format 4: PC float

Format 5: Sun float

Format 6: PC32

Format 7: Sun32

Format 8: PC16 unsigned

Format 9: Sun16 unsigned

See Also: Vclose, Vnew.

Note that 'V' is a special variable in MSHELL, a virtual variable. With this variable you can manipulate an image file which can be as large as the whole disk space available in the system. If the user has a huge image in a file or is going to be working with an image that can not be easily hold in memory, then he or she can still manipulate pieces of the large image using MSHELL's virtual variable.

Once a link is established between a file in disk and the virtual variable 'V', then the user can access rectangular regions of interest in the disk file for read or write operations (the user must always provide a rectangular region of interest when writing or reading from the 'V'). The following example illustrates the use of a virtual variable. It corresponds to the script file **flyby.msh** located in the MSHELL **msh** distribution directory.

```
       M_cwd = "/proview/images/clemen/moonbrus"
roi = wdef(0,0,1,1)
V = Vopen("allmoon.chr",5760::11521::1::0,roi,0);
flyby = 0
```

```
view flyby
i=0
while(i<35){
       meter("flyover virtual image",i/35*100)
       angle = i/35*6.28
       roi = wdef(2336+128*cos(angle), 6926+128*sin(angle),
256,256)
       flyby = V(roi)
       i = i+1
}
meter("",-1)
```

var *Variance*

Syntax: **var**(a) or a.**var**

Description: Given an input array, 'a', **var**(a) computes the variance of the elements in 'a'. The variance is computed using the following expression,

$$\mathrm{var}(a) = \frac{1}{J \cdot I} \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} (a_{j,i} - \bar{a})^2 \quad \text{where,}$$

$\bar{a} = \frac{1}{J \cdot I} \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} (a_{j,i})$ is the sample mean of 'a' ,for all j, i;

where j and i are, respectively, row and column indices.

varname *Returns the Variable Name*

Syntax: **x.varname**

Description: returns the name of the variable as a string, i.e. ‘x’.
For example $mystring = x.varname

See Also: M_inputfocus

vartype Returns the Variable Type

Syntax: **vartype**(variable)

Description: This is used to pass the "type" of the specified variable to the screen so that the user knows what "type" a particular variable is. "Type" is a number which can be one of the following possibilities:

0 = means that the variable does not exist

1 = signifies an integer or array

2 = signifies a string

vec2image Vector Format to Image Format

Syntax: **vec2image()**

Description: converts from vector format into an image representation in memory. (UNDER DEV.)

viewout Closes array viewing window

Syntax: **viewout** image

Description: Closes array viewing window associated with the image variable. (GUI only).

view Enables Screen Display of an Image

Syntax: **view** myimage

Description: From the command line or from a script file, use this command to enable the display of any image variable, i.e. 'myimage', that already exits in memory. Note: after version 3 of MSHELL, you can assign indivudal palettes to the image before it is displayed by setting the ‘x.LUT’ attribute.

Example:

```
x = reada(“eqohare.chr”,”char”);  // open and image
view x;
y = x;        // assign x to y
y.LUT = (255,0,1,3)   // assign an inverse palette to y
view y;
```

view255 Enables Screen Display of a Scaled Image

Syntax: **view255** myimage

view255 myimage,exclude_val_vec

Description: From the command line or from a script file, use this command to enable the display of any image variable, i.e. 'myimage', that already exits in memory. The display image will be set to use autoscaling. In the second usage usage, the user can control the automatic scaling such that a range of values provided by 'exclude_val_vec' are not used in the computation. This is particularly usefull to avoid non-image-data values.

Example:

```
Image=hammiw(256,256);
view255 Image;    // autoscales the image
Image2 = Image;   // make a copy
Image2(0,0)=1/0   // introduce an outliner
Image2(5,5)=100   // introduce another outliner
view255 Image2,1/0::100   // will display a nice image
```

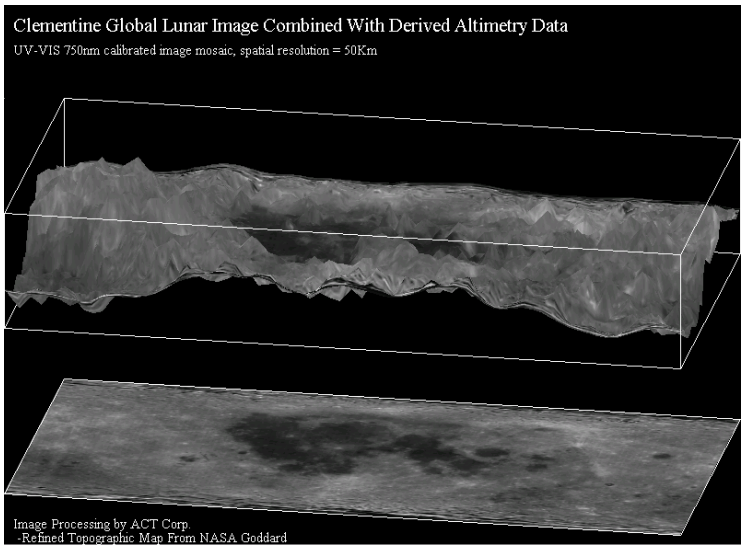
view4d project planar image into an xyz grid

Syntax: **view4d**(image,xy,z,rot::tilt::nout::ns)
Note: this function is being replaced by '**image2surface'**

Description: Given an input image defined over a rectangular grid, this command will reproject the image intensities to a user requested xyz grid. The command is called view4d because it uses amplitude and xyz information (4dimensions). It also applies a rotation of 'rot', and a tilt of 'tilt', generating and output image of 'nout' rows/columns. The sampling of the input xy or z place is controlled via 'ns=1,…,8', where ns=1 is most accurate and ns=8 is the least accurate.

Example 1: The following example uses view4d to take the albedo image of the moon and and a height image of the moon and generate a draped version of the albedo image over the Digital Terrain elevation image. In the script file below, the 'uvvis750.flt' file corresponds to the lower image; in the top image this file has been altitude enhanced with the topography image 'luntopo/4'.

```
M_cwd = M_proviewdir::"/IMAGES"
luntopo = reada("luntopo.flt","float")

uvvis750 = reada("uvvis750.chr","char")
uvvis750 = decimate(uvvis750,2,2)

luntopo = shiftc(luntopo,0,180)

luntopo    = scale255(luntopo)
view luntopo
MOON = view4d(uvvis750,0,luntopo/4,-10::70::800::5)
MOON.m_viewflag=0

MOON = zeropad(MOON,MOON.nrows+100,MOON.ncols)
MOON = shiftc(MOON,100,0)

MOON.m_viewx0=0
MOON.m_viewy0=0
MOON.m_viewwidth=MOON.ncols+50
MOON.m_viewheight=MOON.nrows+50
MOON.m_viewflag=1 // make image visible
view MOON
// *******ADD TEXT TO OUTPUT IMAGE
row=10; col=10
$text = "Clementine Global Lunar Image Combined With Derived
Altimetry Data"
textoverlay(MOON,$text,"Times New
Roman",24::col::row::9::1::1)

row=40; col=10
$text = "UV-VIS 750nm calibrated image mosaic, spatial
resolution = 50Km "
textoverlay(MOON,$text,"Times New
Roman",18::col::row::9::1::1)

row = MOON.nrows-30; col=10
$text = "Image Processing by ACT Corp."
textoverlay(MOON,$text,"Times New
Roman",18::col::row::9::1::1)

row = MOON.nrows-15; col=10
$text = " -Refined Topographic Map From NASA Goddard"
textoverlay(MOON,$text,"Times New
Roman",18::col::row::9::1::1)
```


Clementine Global Lunar Image Combined With Derived Altimetry Data
UV-VIS 750nm calibrated image mosaic, spatial resolution = 50Km
Image Processing by ACT Corp.
-Refined Topographic Map From NASA Goddard

- W -

wclose Closes all Screen Windows

Syntax: **wclose**

Description: Closes all the windows presently opened in the MSHELL environment.

wcolut[#] Color Look up Table

Syntax: **wcolut**[#]

Description: Used to list and define all three rows (red, green, blue) of a color look up table. Use the **select** command or the **m_viewlut** intrinsic with wolut# to choose one of the palettes for viewing.

0=gray

1=inverse gray

2=pseudocolor

3=inverse pseudocolor

4=user defined

Note: starting in MSHELL version 3, the scheme for tracking lookup tables associated with array variables was changed. See array attribute **'x.LUT'** *for details. That is, a color palette can be assigned to each individual array.*

wdef Define a Region of Interest

Syntax: **wdef**(ul_row,ul_col,nrows,ncols)

Description: This function is used to define a region of interest or window, where: 'ul_row' and 'ul_col' are the upper left pixel coordinates of the region of interest; 'nrows' and 'ncols' are, respectively, the number of rows and columns contained in the region of interest.

Example: The following command defines **roi**1 to be a 16 x 16 rectangular region of interest with upper left corner at row = 8 and col = 6.

```
[ready]: roi1 = wdef(8,6,16,16)
[ready]: froi = f(roi1)   // extracts subimage
```

while *While Loop*

Syntax: **while** *(condition)* {

statements

counter

}

Description: This is a type of loop which runs until the condition stated is nolonger met.

wmove *Move a Region of Interest*

Syntax: **wmove**(roi,row,col)

Description: Given a valid rectangular region of interest, 'roi', originally defined using **wdef** this function will generate a new region of interest which is a translated version of the input ROI, where 'row' and 'col' are the row and column coordinates of the upper left of the translated region of interest, and 'roi' is the input region of interest.

wolut[#] *Windows Output Look Up Table*

Syntax: **wolut**[#]

Description: This is used to apply one of the output look-up-tables to a particular image. The wolut[#] system variables are used with the **select** command or **m_viewlut** intrinsic to choose a lookup table to use. For example: select wcolut0 will select the grayscale lookup table. To read or modify the lookup tables RGB values, use the wcolut[#] system variables.

0=gray

1=inverse gray

2=pseudocolor

3=inverse pseudocolor

4=user-defined

*Note: starting in MSHELL version 3, the scheme for tracking lookup tables associated with array variables was changed. See array attribute '***x.LUT'** *for details. That is, a color palette can be assigned to each individual array.*

---

writea Write Array to Disk

Syntax: **writea**("fname",a,"mode")

Description: Write an array, 'a', to file 'fname' on the disk in the indicated 'mode', using any of the supported formats. For a discussion on the modes see description provided in the reada() function.

Note: for the 'char', 'float', 'ascii', and 'double' formats, this function will save x.text attribute if it was intitialized in the array to write to disk.

See Also: reada(), Vopen(), fits_...(), nc_...(), pds_...(), writef()

---

writecolor Writes a Color Image

Syntax: **writecolor**("path/filename.ext", r, g ,b, "image type")

Description: NOTE: THIS FUNCTION IS REPLACED BY writeRGB().
Used to write a color image to disk using the three red(r), green(g), blue(b) image color arrays. The image type string is simply the type of image to be written (*tiff*,jpeg,png*)*.

writeRGBA Writes an RGB or RGBA Color Image

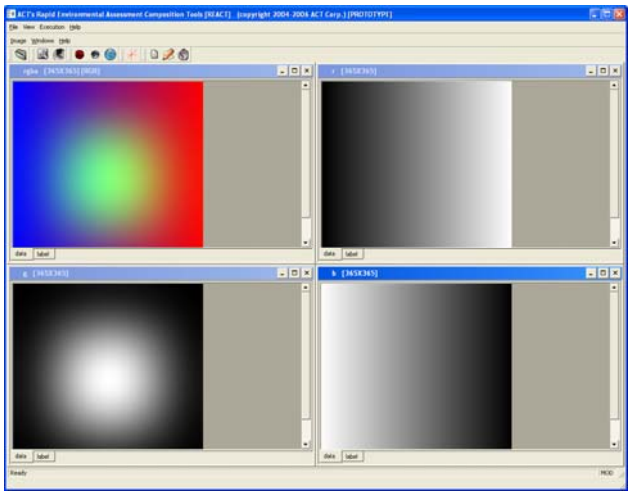
Syntax: **writeRGB**($file, r, g ,b, $type)
writeRGBA($file, r, g ,b, a, $type)

Description: Used to write a color image to disk using the three red(r), green(g), blue(b) image color arrays. The image type string, $type, is simply the type of image to be written, e.g.

BMP, GIF, JPEG, PBM, PGM, PNG, PPM, XBM, XPM

Example:

```
N = 365
r = scale255((0,N-1,1,N))
g = scale255(hammiw(N,N))
b = (255-r)

q = r#g#b
alternateRGB(q);
view q
a = r
writeRGB(M_proviewdir::"/temp/color.png",r,g,b,"PNG");
writeRGBA(M_proviewdir::"/temp/color_rgba.png",r,g,b,a,"PNG")
;
rgba = reada(M_proviewdir::"/temp/color_rgba.png","all");
view rgba
```



writef *Formatted File Write*

Syntax: **writef**(unit,"format",arrayname) or

writef(unit,"format",stringname)

Description: This function is used to perform a formatted write to a file unit which has already being open using the **openf** command. Note that only one value or string can be written at a time**.** The first command above is used for doing a formatted write of an already existing 1 x 1 array into a file. The second command above is used for doing a formatted string write into an already open file. The format string follows a similar form to the 'C' language formatting options, where %s is used for strings, and %f, %g, %e are used for floating point numbers.

Example: Given that unit 1 has been already open with an **openf** statement, and that 'x' is an array variable and 'name' is a string variable, then the following MSHELL statements are legal **writef** statements,

```
       status=writef(1,"Hello my name is %s \n",name);
status=writef(1,"The temperature is %f degrees",x);
status=writef(1,"The value is \n %4.2f",x);
status=writef(1,"%g",x);
status=writef(1,"%e",x); .
```

wsize *Size of a Region of Interest*

Syntax: **wsize**(roi) or roi.**wsize**

Description: Given a valid rectangular region of interest, this function will return the dimensions of the rectangular window defining the ROI.

wtile *Tiles all Screen Windows*

Syntax: **wtile**

Description: Tiles all the windows presently opened in the MSHELL environment.

- X -

xcorr Cross Correlation

Syntax: **xcorr**(a,b)

Description: Perform the cross-correlation between two input arrays. There are no specific restrictions on the two input arrays: they can be either real or complex and one or two dimensional. If 'a' has dimensions (N x M), and 'b' has dimensions (P x Q) the resulting ARRAY will have dimensions (N+P-1, M+Q-1). The implementation used in 'xcorr' is computationally efficient for small arrays. For large array sizes, an **FFT** implementation should be considered.

See Also: **xcorrt**

Example: Example with **xcorrt.**

xcorrfft Cross-correlation of two FFT's

Syntax: **xcorrfft**(FFT1, FFT2)

Description: This is used to find the cross-corrleation between two computed FFT's of certain previous arrays. This command is especially advantageous where a cross-correlation of two large arrays would be processor comsumptive; therefore, an FFT (being much smaller) would be much faster to compare.

xcorrt Truncated Cross Correlation

Syntax: **xcorrt**(a,b)

Description: This function is similar to **xcorr** except that it only evaluates the cross correlation in the range of the second array. This function, **xcorr**t, truncates the cross-correlation results by only evaluating the cross-correlation over the range of 'b'. Note that 'a' is assumed to have odd dimensions.

Example: The following illustrates the use of **xcorr** and **xcorrt**,

```
[ready]: x = 2::4::2
[ready]: y = (0,2,1)
[ready]: z = xcorr(x,y)       // example of xcorr
row 0 =
  0.00   2.00   8.00  10.00   4.00
[ready]: zp = xcorrt(x,y)     // example of xcorrt
[ready]: zp
row 0 =
8.00  10.00
```

xline Extract Pixel Values along Line Segment

Syntax: **xline**(a,row1,col1,row2,col2)

Description: This function extracts the pixel values along a line segment in an array. The coordinates (row1, col1) and (row2, col2) are, respectively, the start and end points of the line segment. Note that **xline**(a,row1,col1,row2,col2) is equivalent to the following MSHELL instruction based on the **bresen** and **complex** unctions: **bresen**(**complex**(col1::col2,row1::row2)).

See Also: **bresen.**

Example: For the given an array, the following will calculate both the xline and the a(besen(complex(0::3, 0::3))) transformations of the array.

```
A = hammiw(4,4)               // generate a 4 x 4 array
a
 (10^0) X
row 0 =
    0.0064     0.0432     0.0800     0.0432
row 1 =
    0.0432     0.2916     0.5400     0.2916
row 2 =
    0.0800     0.5400     1.0000     0.5400
row 3 =
    0.0432     0.2916     0.5400     0.2916
xline(a,0, 0, 3 ,3)              // example of xline
 (10^0) X
row 0 =
    0.0064       0.2916     1.0000     0.2916
0.0432
a(bresen(complex(0::3,0::3)))
    0.0064       0.2916     1.0000     0.2916
```

xlinec Extracts Coordinates of Line

Syntax: **xlinec**(image)

Description: Used to list the coordinates along a linear region of interest. Once the command has been issued in the command window, then one is prompted to click the endpoints of the desired linear region within the image of interest.

xlinev Extracts Vertices of Line

Syntax: **xlinev**(image)

Description: Used to list the vertices or end point of a linear region of interest. Once the command has been issued in the command window, then one is prompted to click the endpoints of the desired linear region within the image of interest.

xlut Look-Up-Table Transformation

Syntax: **xlut**(a,lut)

Description: Performs a permanent look up table transformation on the given input array 'a' using a user supplied look-up-table, 'lut'.

Example: Suppose it is desired to transform an array region of interest in image f using an inverse ramp mapping. This can be done using,

```
f = reada("eqohare.chr","char")
roi = wdef(0,0,20,30)
f(roi) = xlut(  f(roi) , (255,0,1) ) ;
```

xmlattrvalues Extracts the attribute values

Syntax: **xmlattrvalues("keyword","value")**

Description: this function is used to extract the attribute values associated with tags that share the same name. The input arguments are the tag name and the attribute name.

Example:

```
[ready]: //tag name: "keyword", attribute name:"value"
[ready]: $attr = xmlattrvalues("keyword","value")
```

xmlparse parse an XML string.

Syntax: **xmlparse($meta)**

Description: This function is used to parse an XML string. When excuted it loads dynamically (behind the scene) msh32xml.dll. The input argument is an XML string. It returns 1 if the parsing was successful l, otherwise it returns 0.

Example:

```
[ready]: status = xmlparse($meta) //$meta is an XML string
[ready]:
```

xmltextvalue Extracts the text value of a

Syntax: **xmltextvalue("metadata.idinfo.spdom.bounding.northbc")**

Description: This function is used to extract the text values of a particular node in the DOM tree. The input argument is a string where the node names in the hierarchy are separated by '.'.

Example:

```
[ready]: $nbc =
xmltextvalue("metadata.idinfo.spdom.bounding.northbc")
[ready]:
```

xmltextvalues Extracts the text values

Syntax: **xmltextvalues("northbc")**

Description: This function is used to extract the text values associated with tags that share the same name. The input argument is a string containing the tag name.

Example:

```
[ready]: $txtvals = xmltextvalues("northbc")
[ready]:
```

xpolyc Extracts Coordinates of a Polygon

Syntax: **xpolyc**(image)

Description: Used to list the coordinates along a polygonal region of interest. Once the command has been issued in the command window, then one is prompted to click the vertices of the desired polygonal region within the image of interest.

---

xpolyv *Extracts Vertices of a Polygon*

Syntax: **xpolyv**(image)

Description: Used to list the vertices of a polygonal region of interest. Once the command has been issued in the command window, then one is prompted to click the vertices of the desired polygonal region within the image of interest.

- Z -

zeropad *Expand an Image with Zeroes*

Syntax: **zeropad**(a,n,m)

Description: Add zeros to the input array 'a', where n is the number of rows and m is the number of columns.

Example: Takes a 2 x 2 array and pads it to a 3 x 5 array.

```
[ready]: x = (1::2)#(3::4)
[ready]: x
row 0 =
  1.00   2.00
row 1 =
4.00
[ready]: y = zeropad(x,3,5)
[ready]: y
row 0 =
  1.00   2.00   0.00   0.00   0.00
row 1 =
  3.00   4.00   0.00   0.00   0.00
row 2 =
  0.00   0.00   0.00   0.00   0.00
```

zeros *Initialize Array to all Zeros*

Syntax: **zeros**(n,m)

Description: Create an array in memory with all the elements set to 0.

Example: The following MSHELL statement will create the 512 x 512 array 'c' in memory with all entries set to 0.

```
C = zeros(512,512);
```

zinterp *Zero Order Interpolation*

Syntax: **zinterp**(f,x)

Description: This is used to perform zero-order interpolation on an array “f”, by expanding the data to a range specified by the “x” array which will list the abscissa and ordinate indices for each new desired element.

Appendix C: Remote Access

Introduction

A new feature **under development** is the communication between two MSHELL engines located in different machines. This is done via CORBA. The following table contains the present set of function calls that are being tested and that have been added to the MSHELL language.

The following internal functions (defined in Appendix B) support remote invocations of MSHELL (using CORBA):

rs_init
rs_connect
rs_release
rs_exec
rs_exec_async
rs_wait
rs_getString
rs_getArray
rs_availWipeServers
rs_availDataSets
rs_getMetaData
rs_extractData

See detail discussion of the above functions in Appendix B.

Appendix D: Cartographic Library

PROJ CopyRight Notice

The MSHELL function Geo2Cart is based on the PROJ cartographic program/library developed by USGS. Copyright to PROJ and usage description follows.

CopyRight Notice associated to the Cartographic DLL

All source, data files and other contents of the PROJ.4 package are

available under the following terms. Note that the PROJ 4.3 and earlier

was "public domain" as is common with US government work, but apparently

this is not a well defined legal term in many countries. I am placing

everything under the following MIT style license because I believe it is

effectively the same as public domain, allowing anyone to use the code as

they wish, including making proprietary derivatives.

Though I have put my own name as copyright holder, I don't mean to imply

I did the work. Essentially all work was done by Gerald Evenden.

Copyright (c) 2000, Frank Warmerdam

Permission is hereby granted, free of charge, to any person obtaining a

copy of this software and associated documentation files (the "Software"),

to deal in the Software without restriction, including without limitation

the rights to use, copy, modify, merge, publish, distribute, sublicense,

and/or sell copies of the Software, and to permit persons to whom the

Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included

in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS

OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL

THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING

FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER

DEALINGS IN THE SOFTWARE.

NAME

proj - forward cartographic projection filter
invproj - inverse cartographic projection filter

SYNOPSIS

proj [**-bceEfiIlmorsStTvVwW** [*args*]] [**+args**] file[s]
invproj [**-bceEfiIlmorsStTwW** [*args*]] [**+args**] file[s]

DESCRIPTION

Proj and *invproj* perform respective forward and inverse transformation of cartographic data to or from cartesian data with a wide range of selectable projection functions.
The following control parameters can appear in any order:

-b

Special option for binary coordinate data input and output through standard input and standard output. Data is assumed to be in system type *double* floating point words. This option is to be used when **proj** is a *son* process and allows bypassing formatting operations.

-i

Selects binary input only (see **-b option).**

-I

alternate method to specify inverse projection. Redundant when used with **invproj.**

-o

Selects binary output only (see **-b option).**

-t*a*

A specifies a character employed as the first character to denote a control line to be passed through without processing. This option applicable to ascii input only. (# is the default value).

-e *string*

String is an arbitrary string to be output if an error is detected during data transformations. The default value is: *\t*. Note that if the **-b, -i** or **-o** options are employed, an error is returned as HUGE_VAL value for both return values.

-E

causes the input coordinates to be copied to the output line prior to printing the converted values.

-l*[p|P|=|e|u]***id**

List projection identifiers with **-l, -lp** or **-lP (expanded)** that can be selected with **+proj. -l=***id* gives expanded description of projection *id.* List ellipsoid identifiers with **-le,** that can be selected with **+ellps** or **-lu** list of cartesian to meter conversion factors that can be selected with **+units.**

-r

This options reverses the order of the expected input from longitude-latitude or x-y to latitude-longitude or y-x.

-s

This options reverses the order of the output from x-y or longitude-latitude to y-x or latitude-longitude.

-S

Causes estimation of *meridinal* and *parallel* scale factors, *area* scale factor and *angular distortion,* and *maximum* and *minimum* scale factors to be listed between <> for each input point. For conformal projections meridinal and parallel scales factors will be equal and angular distortion zero. Equal area projections will have an area factor of 1.

-m *mult*

The cartesian data may be scaled by the *mult* parameter. When processing data in a forward projection mode the cartesian output values are multiplied by *mult* otherwise the input cartesian values are divided by *mult* before inverse projection. If the first two characters of *mult* are 1/ or 1: then the reciprocal value of *mult* is employed.

-f *format*

Format is a *printf* format string to control the form of the output values. For inverse projections, the output will be in degrees when this option is employed. If a format is specified for inverse projection the output data will be in decimal degrees. The default format is "%.2f" for forward projection and DMS for inverse.

-[w|W]*n*

N is the number of significant fractional digits to employ for seconds output (when the option is not specified, **-w3** is assumed). When **-W** is employed the fields will be constant width and with leading zeroes.

-v

causes a listing of cartographic control parameters tested for and used by the program to be printed prior to input data. Should not be used with the **-T** option.

-V

This option causes an expanded annotated listing of the characteristics of the projected point. **-v is implied with this option.**

-T *ulow,uhi,vlow,vhi,res[,umax,vmax]*

This option creates a set of bivariate Chebyshev polynomial coefficients that approximate the selected cartographic projection on *stdout.* The values *low* and *hi* denote the range of the input where the *u* or *v* prefixes apply to respective longitude-x or latitude-y depending upon whether a forward or inverse projection is selected. *Res* is an integer number specifying the power of 10 precision of the approximation. For example, a *res* of -3 specifies an approximation with an accuracy better than .001. *Umax,* and *vmax* specify maximum degree of the polynomials (default: 15). See also: fproj**(1).**

The **+args** run-line arguments are associated with cartographic parameters and usage varies with projection and for a complete description see *Cartographic Projection Procedures for the UNIX Environment---A User's Manual)* and supplementary documentation for Release 4.

Additional projection control parameters may be contained in two auxilliary control files: the first is optionally referenced with the **+init=***file:id* and the second is always processed after the name of the projection has been established from either the run-line or the contents of **+init** file. The environment parameter **PROJ_LIB** establishes the default directory for a file reference without an absolute path.

One or more *files* (processed in left to right order) specify the source of data to be transformed. A - will specify the location of processing standard input. If no files are specified, the input is assumed to be from *stdin.* For ASCII input data the two data values must be in the first two white space separated fields and when both input and output are ASCII all trailing portions of the input line are appended to the output line.

Input geographic data (longitude and latitude) must be in DMS format and input cartesian data must be in units consistent with the ellipsoid major axis or sphere radius units. Output geographic coordinates will be in DMS (if the **-w** switch is not employed) and precise to 0.001" with trailing, zero-valued minute-second fields deleted.

EXAMPLE Usage

The following script

```
proj +proj=utm +lon_0=112w -r <<EOF
 45d15'33.1"    111.5W

 45d15.551666667N       -111d30

 +45.25919444444        111d30'000w

 EOF
```

will perform UTM forward projection with a standard UTM central meridian nearest longitude 112°W. The geographic values of this example are equivalent and meant as examples of various forms of DMS input. The x-y output data will appear as three lines of:

```
460769.27      5011648.45
```

SEE ALSO

Cartographic Projection Procedures for the UNIX Environment---A User's Manual, (Evenden, 1990, Open-file report 90-284).
Map Projections Used by the U. S. Geological Survey (Snyder, 1984, USGS Bulletin 1532).
Map Projections---A Working Manual (Synder, 1988, USGS Prof. Paper 1395).
An Album of Map Projections (Snyder & Voxland, 1989, USGS Prof. Paper 1453).

HOME PAGE FOR THIS CODE

http://www.remotesensing.org/proj

Appendix E: Geographic Data Abstraction Library

GDAL CopyRight Notice

is a translator library for raster geospatial data formats that is released under an **X/MIT** style **Open Source** license. As a library, it presents a **single abstract data model** to the calling application for all supported formats. The related **OGR** library (which lives within the GDAL source tree) provides a similar capability for simple features vector data. More information can be found at: **http://www.remotesensing.org/gdal/index.html**

GDAL is under copyright by Frank Warmerdam. Copyright notice follows:

Copyright (c) 2000, Frank Warmerdam

Permission is hereby granted, free of charge, to any person obtaining a

copy of this software and associated documentation files (the "Software"),

to deal in the Software without restriction, including without limitation

the rights to use, copy, modify, merge, publish, distribute, sublicense,

and/or sell copies of the Software, and to permit persons to whom the

Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included

in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS

OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL

THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING

FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

REACT/MSHELL-GDAL Interface

REACT/MSHELL has incorporated support to the **open-source G**eographic **D**ata **A**bstraction **L**ibrary (**GDAL**). REACT/MSHELL mostly leverages on GDAL to facilitate the access to many geographical raster-data-formats, see Table 2, and to read and convert widely used spatial reference systems (**SRS**) representations in the open-source community (proj4 and WKT formats).

A number of functions in the MSHELL scripts tree subdirectory use of the GDAL low level functions. In particular the cartographic map projection script functions. i.e.

map_image.msf - cartographic map of a single image file
map_images.msf - cartographic mosaic of a list of image files
map_image_to_file.msf - creates cartographic mosaic on disk (good for HUGE mosaics)

which are located in the …\Proview\SCRIPTS\Satellite_Image_Mapping\Mosaics directory.

These script function are by themselves great examples, within MSHELL, on the use of the low level GDAL functions. The above script functions can handle input files that use ground control points (GCP) for geo-referencing, or that are already in cartographic projection space.

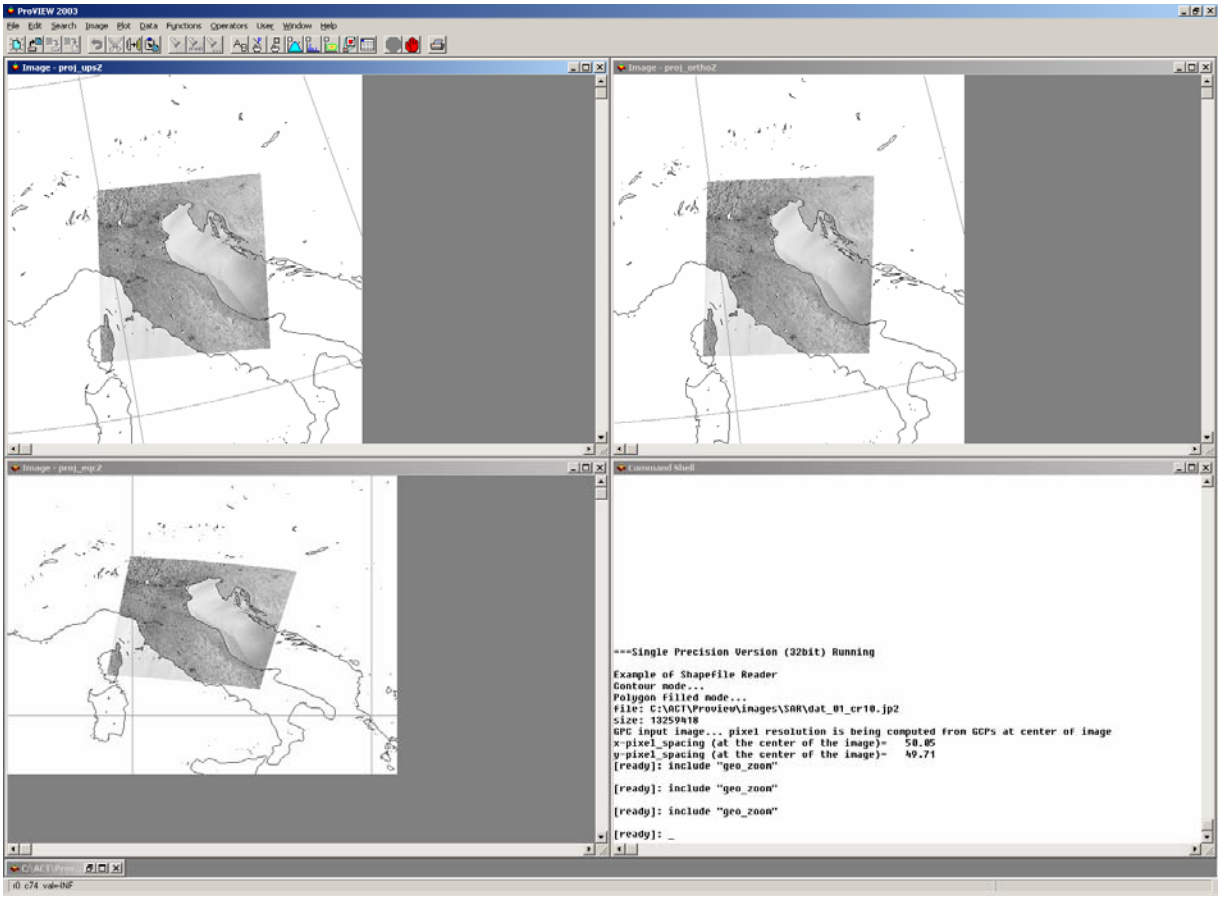

ProVIEW 2003
Image - proj_upsZ
Image - proj_orthoZ
Image - proj_eqcZ
Command Shell
===Single Precision Version (32bit) Running
Example of ShapeFile Reader
Contour mode...
Polygon filled mode...
file: C:\ACT\Proview\images\SAR\dat_01_cr10.jp2
size: 13259x18
GPC input image... pixel resolution is being computed from GCPs at center of image
x-pixel_spacing (at the center of the image)= 50.05
y-pixel_spacing (at the center of the image)= 49.71
[ready]: include "geo_zoom"
[ready]: include "geo_zoom"
[ready]: include "geo_zoom"
[ready]: _

Figure 7 Map_Image.msf Sample ouput screen showing results in

REACT/MSHELL GDAL Library Support

$driver_list = **gdal_driver_init**()

Initializes the GDAL dataset drivers. Returns a list of registered dataset drivers. This must be done before accessing any of the GDAL related functions.

dataset_handle = **gdal_open**($dataset_name) or **gdal_open**($data_name,RWMODE)

Attempts to open a dataset with GDAL library and returns a non-zero dataset handle (integer number) which is used to reference that file in other functions. (in many cases *$dataset_name* is just a file name), RWMODE=0 for opening on read only mode, and RWMODE=1 for opening using write/update mode.

gdal_close(dataset_handle)

Closes the dataset.

gdal_dataset_info(dataset_handle)

Returns some general information about the given open dataset_handle.

gdal_dataset_info_detail(dataset_handle)

Returns detail information about the given open dataset_handle.

gdal_get_metadata(dataset_handle,"") or **gdal_get_metadata**(dataset_handle,"SUBDATASETS")

Returns metadata information

gdal_get_raster_size(dataset_handle)
Returns the raster size as a vector array: *bands::rows::columns.*

geo_xform = **gdal_get_geo_transform**(dataset_handle)

Returns the affine transformation coefficients as a vector array. These coefficients satisfy the following equations which transform between pixel/line (P,L) raster space, and projection coordinates (Xp,Yp) space:

```
P – pixel
L = line
Xp = geo_xform(0,0) + P* geo_xform(0,1) + L* geo_xform(0,2)
Yp = geo_xform(0,3) + P* geo_xform(0,4) + L* geo_xform(0,5)
```

gdal_set_geo_transform(dataset_handle, geo_xform**)**

Sets the affine transformation coefficients from a vector array. These coefficients satisfy the following equations which transform between pixel/line (P,L) raster space, and projection coordinates (Xp,Yp) space:

```
P – pixel
L = line
Xp = geo_xform(0,0) + P* geo_xform(0,1) + L* geo_xform(0,2)
Yp = geo_xform(0,3) + P* geo_xform(0,4) + L* geo_xform(0,5)
```

$proj_ref = **gdal_get_projection_ref**(hdl)

Returns the projection definition string for the dataset. The return string defines the projection coordinate system of the image in OpenGIS WKT format. It should be suitable for use with the OGRSpatialReference functions.

status = **gdal_set_projection**(hdl, $proj_ref)

Set the projection reference string for this dataset.

The string should be in OGC WKT or PROJ.4 format. An error may occur because of incorrectly specified projection strings, because the dataset is not writable, or because the dataset does not support the indicated projection. Many formats do not support writing projections. Some formats do not support PROJ.4 strings directly. Convert to WKT with OGRSpatialReference functions first.

gcps = **gdal_get_gcps**(hdl)

Returns the internal Ground Control Points (GCPs) for the dataset as a Nx5 array where N is the number of GCPs, and the 5 columns are arranged as follows:

line::pixel::X::Y::Z

gdal_set_gcps(hdl,geo_xform,$WKT) ?????

Sets the ground control points (GCPs). ‘geo_xform’ contains the geo-transformation, and ‘$WKT’ is the as string with the OGC Well-lknown-Text, that provides the spatial-reference-system to be associated with the ground control points. ??????

$gcp_proj = **gdal_get_gcp_projection**(hdl)

Returns the projection space (a WKT string) for the X,Y,Z values returned from *gdal_get_gcps().*

count = **gdal_get_overview_count**(hdl,band)

Returns the number of overview layers available.

ctable = **gdal_get_raster_color_table**(hdl,bandnum)

Returns an array with the color table.

ndatav = **gdal_get_raster_no_data_value**(hdl,bandnum)

Returns the ‘no-data-value’ for this band number.

offset = **gdal_get_raster_offset**(hdl,bandnum)

Fetch the raster value offset.

This value (in combination with the **gdal_get_raster_scale**( ) value) is used to transform raw pixel values into the units returned by gdal_get_raster_unit_type().

Units value = (raw value * scale) + offset

For file formats that don't know this intrinsically a value of zero is returned.

ctable = **gdal_get_raster_data_type**(hdl,bandnum)

Returns the pixel data type for this band. See gdal_create( ) for a description of the data types.

ctable = **gdal_get_raster_scale**(hdl,bandnum)

Fetch the raster value scale.

This value (in combination with the **gdal_get_raster_offset**( ) value) is used to transform raw pixel values into the units returned by gdal_get_raster_unit_type().

Units value = (raw value * scale) + offset

For file formats that don't know this intrinsically a value of zero is returned.

$unit = **gdal_get_raster_unit_type**(hdl,bandnum)

Returns a string with the unit type for this raster band.

gdal_rasterio(hdl, wf, xoff, yoff, xsize, ysize, my_array, band_map, npix_spc, nline_spc, nband_spc)

hdl	handle for an open dataset
wf	write flag (1 for write, 0 for read)
xoff	raster x offset
yoff	raster y offset
xsize	pixel width to read/write
ysize	pixel height to read/write
my_array	array to read or write
band_map	vector array of band numbers to read/write
npix_spc	interpixel spacing in bytes (0 = default, datatype size)
nline_spc	interline spacing in bytes (0 = default, datatype size * buffer width)
nband_spc	interband spacing in bytes (0 = default, nline_spc * (buffer height/nbands requested))

If xsize or ysize is different than the size of *my_array*, the GDAL library will decimate, or replicate to fit the *my_array* buffer.

The float (or double) data in my_array will be converted (just clipping and casting, values not scaled!) to whatever data type the dataset is defined to be.

new_hdl = **gdal_create_copy**($driver_name,$filename,hdl, options)

Creates a new GDAL dataset in *$filename* using existing open dataset referenced by *hdl*.
The *$driver_name* is the “short” name for the dataset format, which are shown in the output of the *gdal_driver_info()* function.
The *options* are for future general, and format specific options.
Returns a new dataset handle (may be read-only)

hdl_new = **gdal_create**($driver_name,$filename,size_vector,datatype_code,0)

Creates a new GDAL dataset from scratch.
The *$driver_name* is the "short" name for the dataset format, which are shown in the output of the *gdal_driver_info()* function.
The *size_vector* should always be 3 values given as: *bands::rows::columns*.
The *datatype_code* is the pixel datatype code used by GDAL:

```
GDT_Byte = 1,
GDT_UInt16 = 2,
GDT_Int16 = 3,
GDT_UInt32 = 4,
GDT_Int32 = 5,
GDT_Float32 = 6,
GDT_Float64 = 7,
GDT_CInt16 = 8,
GDT_CInt32 = 9,
GDT_CFloat32 = 10,
GDT_CFloat64 = 11
```

status = **gdal_build_overviews**(hdl,$resampling_method,overview_list,band_list)

Creates overviews for the dataset if the driver for that dataset supports them.

hdl – dataset handle
$resampling_method – “AVERAGE”, “NEAREST”, …others in future?
overview_list – vector of reduction factors to use creating overviews
band_list – vector of bands to generate overviews for (band #’s start at 1)

status = gdal_contour_generate (hdl, band,contour_interval,contour_base, fixed_level_count,fixed_levels,use_nodata,$shapefile,$attr_name,use_3d)

hdl	handle for an open dataset
contour_interval	The elevation interval between contours generated.
contour_base	The "base" relative to which contour intervals are applied. This is normally zero, but could be different. To generate 10m contours at 5, 15, 25, ... the ContourBase would be 5.
fixed_level_count	The number of fixed levels. If this is greater than zero, then fixed levels will be used, and ContourInterval and ContourBase are ignored.
fixed_levels	The list of fixed contour levels at which contours should be generated. It will contain FixedLevelCount entries, and may be NULL if fixed levels are disabled (FixedLevelCount = 0).
use_nodata	If this is 1, and if the dataset band has a nodata value associated with it, the nodata value will be honoured. That is, no contours will be generated for nodata pixels.
$shapefile	The name of the shapefile to generate. The name does not require an extension (the .shp, .shx, .dbf, and .prj extensions will be added as needed). If the shapefile already exists, it will be deleted and a new one created.
$attr_name	The name of the attribute that should be created in the shapefile and used for elevations. This may be NULL if no elevation attribute is required.
use_3d	1 if the produced shapefile should be 3D, with the contour elevation associate with each vertex in the file. Note that 3D shapefiles are not nearly as widely supported by applications as 2D shapefiles.

Example: see example in function shape2contour.

REACT/MSHELL OGR (within GDAL) Library Support

osr_hdl = **osr_new_spatial_reference**($WKT)

Creates a new OpenGIS Spatial Reference System (**SRS**) object and returns a handle to this object. The input $WKT is optional. An empty string can be passed.

status = **osr_destroy_spatial_reference**(osr_hdl)

Destroys the SRS object.

status = **osr_import_from_proj4**(osr_hdl, $proj4)

Import PROJ.4 coordinate string. The OpenGIS Spatial Reference object is initialized from the passed PROJ.4 style coordinate system string.

Example: pszProj4 = "+proj=utm +zone=11 +datum=WGS84"

$WKT = **osr_export_to_wkt**(osr_hdl)

Returns the WKT string for this SRS.

$proj4 = **osr_export_to_proj4**(osr_hdl)

Returns the PROJ.4 string for this SRS.

Table 2 GDAL Supported Raster Formats (see http://www.remotesensing.org/gdal/formats_list.html updates)

Long Format Name	Code	Creation	Georeferencing	Accessbile From MSHELL	Maximum file size[1]
Arc/Info ASCII Grid	AAIGrid	Yes	Yes		No limits
Arc/Info Binary Grid (.adf)	AIG	No	Yes		--
Microsoft Windows Device Independent Bitmap (.bmp)	BMP	Yes	Yes		4GiB
BSB Nautical Chart Format (.kap)	BSB	No	Yes		--
CEOS (Spot for instance)	CEOS	No	No		--
First Generation USGS DOQ (.doq)	DOQ1	No	Yes		--
New Labelled USGS DOQ (.doq)	DOQ2	No	Yes		--
Military Elevation Data (.dt0, .dt1)	DTED	No	Yes		--
ERMapper Compressed Wavelets (.ecw)	ECW	Yes	Yes		
ESRI .hdr Labelled	EHdr	No	Yes		--
ENVI .hdr Labelled Raster	ENVI	Yes	Yes		No limits
Envisat Image Product (.n1)	Envisat	No	No		--
EOSAT FAST Format	FAST	No	Yes		--
FITS (.fits)	FITS	Yes	No		
Graphics Interchange Format (.gif)	GIF	Yes	No		
Arc/Info Binary Grid (.adf)	GIO	Yes	Yes		
GRASS Rasters	GRASS	No	Yes		--
TIFF / GeoTIFF (.tif)	GTiff	Yes	Yes		4GiB
Hierarchical Data Format Release 4 (HDF4)	HDF4	Yes	Yes		2GiB
Erdas Imagine (.img)	HFA	Yes	Yes		No limits[2]
Atlantis MFF2e	HKV	Yes	Yes		No limits
Japanese DEM (.mem)	JDEM	No	Yes		--
JPEG JFIF (.jpg)	JPEG	Yes	Yes		4GiB (max dimentions 65500x65500)
JPEG2000 (.jp2, .j2k)	JPEG2000	Yes	Yes		
JPEG2000 (.jp2, .j2k)	JP2KAK	Yes	Yes		
NOAA Polar Orbiter Level 1b Data Set (AVHRR)	L1B	No	Yes		--
Atlantis MFF	MFF	Yes	Yes		No limits
Multi-resolution Seamless Image Database	MrSID	No	Yes		--
NITF	NITF	Yes	Yes		
OGDI Bridge	OGDI	No	Yes		--
PCI .aux Labelled	PAux	Yes	No		No limits
Portable Network Graphics (.png)	PNG	Yes	No		
Netpbm (.ppm,.pgm)	PNM	Yes	No		No limits
USGS SDTS DEM (*CATD.DDF)	SDTS	No	Yes		--
SAR CEOS	SAR_CEOS	No	Yes		--
USGS ASCII DEM (.dem)	USGSDEM	No	Yes		--
X11 Pixmap (.xpm)	XPM	Yes	No		

[1]Maximum file size does not only determined by the file format itself, but operating system/file system capabilities as well. Look **here** for details.
[2]ERDAS Imagine has different file format for large files, where 32-bit pointers cannot be used. Look for details **here**.

Appendix F: RECENT CHANGES

List of Capabilities recently added to REACT/MSHELL

- Added table of index in the manual to all internal function. The index is by category of functions!
- Added **XML** parser
- 9/8/2002 - added M_no_data system variable
- 12/2002 – added M_logfile, and M_logmode
- Improved interface to NetCDF library
- Added interface to the CFITS library
- Added support for **double precision version** of MSHELL.
- Added support for **OpenGL**.
- Added interface to the USGS projection library (**PROJ4**)
- 5/2003 – Added **regulars expressions** capability, similar to Perl syntax
- Enhanced documentation with embedded examples
- Improved ODBC retrieval speed
- 2/2003 - **Ported MSHELL Interpreter to LINUX!**
- 3/2003 - Added M_processid system variable
- 4/2003 – Added support to writing geotiff files via writea()

- 5/2003 – added additional support to dworld such that vector shoreline can now be drawn on any supported USGS cartographic map projections.

- 5/2003 – added support to the KAKADU **JPEG2000 reader and writer**

- 6/2003 – Added support to the Geographic Data Abstraction Library (GDAL)
  ➔ **read/write support to:**
  GeoTiff, NITF, ERDAS, HDF, JPEG2000, NITF, CEOS,ENVISAT, …

- Incorporated **OpenGIS WKT** cartographic description methods

- 10/2003 – writea and reada will save and restore the x.text attribute

- 10/2003 – added support to perform raster two Shape-files conversion on large input files

- 10/2003 - added 'statsx' function, to allow statistics computation with exclusion of certain input values

- 11/11/2003 - added dbrsgetcolnames() function and improved database access

- 12/2004Added new set of mapping scripts.
  Map_Image.msf, Map_Images.msf, Map_Image_to_file.msf, …
  These functions can map project any GDAL supported input file.

- 2/13/04- added support for formatted 'float2str', i.e. float2str(x,$format)

- 2/17/2004- added 'stop_process($error_string)' function

- 3/22/2004- ladd2groi(a,groi,b) can support 'b' to be a row vector, very useful for implementing fast accumulators

- 3/2004 - added additional support for **CFITS** library

- 4/2004 - added capability to MSHELL to be under the control of a license
  server with floating licenses

- 4/2004 fixed documentation for functions menusel and iboxlist

- 7/2004 added example for function gdal_contour_generate

- 1/2005 added a number of additional function bindings between MSHELL and GDAL

- 2/2005 fixed documentation of view255

- 2/2005 added documentation for the 'x.LUT' image attribute which control image palettes.

- 3/2005 added support for a new raster reading function call, x = reada($input,"all"). This function tries to determine the input file type and read it. See also the new M_maxdim system variable.

- 4/2005 added new string manipulation functions, strEndsWith(), strStartsWith(), strContains().

- 4/2005 added volumetric handling capabilities for variables loaded in memory. See 'x.m_voltype', 'x.m_voldim', and the function 'alternateRGB()' plus 'alternateBSQ()'

- 4/2005 added documentation of gdal_get_metadata()
- 4/2005 added information for reading of voldim information, i.e. x.m_nrows3d, x.m_ncols3d, x.n_nband3d.
- 4/2005 added to interpreseter: strContains(), strEndsWith(), strStartWith(), strUniqueList… type of string functions (NOT documented yet)
- 5/2005 added function writeRGB($fname,r,g,b,$format), where $format can be “PNG” or “JPEG”. This function is using a more portable approach to implement similar functionality as in writecolor().
- 5/2005 added x.mapinfo. It is similar to x.text, but it is used to track map_information.
- 7/2005 added maskofLT(),maskofGT, maskofGE(), maskofLE(), maskofEQ()
- 8/2005 added ftp_...., rw…., rmshell…. , getMyIP network related functions
- 8.2005 added sqlite_... , support
- 2.2005 added curvTraj(…) function.
- 6/2006 added support for bsq array IO, e.g. xyz(: | band) = [const or array]
- 3/2007 added readtext_and_status() function. Allows to control flow if the event that the readtext() fails.
- 1/2007 Added variable M_execution_mode. It allows to know which environment the script is running under. Provides more information than M_OS.
- 5/2007 Added new functions: allocate_array, url_copy_to_file, mshift_u16, mshift_u32, and, react_get_carto_info, react_set_carto_map_info.
- 9/2007 Added new functions: react_load_mlt, react_update_layers, react_enable_view, react_get_property, react_get_properties, react_set_property, polygonorient, sys_cat, react_open_url